NASA CONTRACTOR REPORT

NASA CR-2296

DIALOG: AN EXECUTIVE COMPUTER PROGRAM FOR LINKING INDEPENDENT PROGRAMS

by C. R. Glatt, D. S. Hague, and D. A. Watson

Prepared by
AEROPHYSICS RESEARCH CORPORATION
Hampton, Va. 23666
for Langley Research Center

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION • WASHINGTON, D. C. • SEPTEMBER 1973

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No
NASA CR-2296		

4 Title and Subtitle	5 Report Date
DIALOG: AN EXECUTIVE COMPUTER PROGRAM FOR LINKING INDEPENDENT PROGRAMS	September 1973
	6 Performing Organization Code
7. Author(s)	8. Performing Organization Report No
C. R. Glatt, D. S. Hague, and D. A. Watson	
	10. Work Unit No.
9 Performing Organization Name and Address	
AEROPHYSICS RESEARCH CORPORATION P. O. Box 7007 Hampton, Virginia 23666	11. Contract or Grant No. NAS1-10692
	13. Type of Report and Period Covered
12 Sponsoring Agency Name and Address	Contractor Report
National Aeronautics and Space Administration Washington, D.C. 20546	14. Sponsoring Agency Code

15 Supplementary Notes

This is one of two final reports.

16. Abstract

A very large scale computer programming procedure called the DIALOG Executive System has been developed for the CDC 6000 series computers. The executive computer program, DIALOG, controls the sequence of execution and data management function for a library of independent computer programs. Communication of common information is accomplished by DIALOG through a dynamically constructed and maintained data base of common information.

The unique feature of the DIALOG executive system is the manner in which computer programs are linked. Each program maintains its individual identity and as such is unaware of its contribution to the large scale program. This feature makes any computer program a candidate for use with the DIALOG executive system. This manuscript describes the installation and use of the DIALOG Executive System at Langley Research Center. Installation on other CDC 6000 series computers would be similar.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Design, synthesis, ODIN, multiple programming, data base, language, optimization	Unclassified - Unlimited

19 Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	141	$3.00

*For sale by the National Technical Information Service, Springfield, Virginia 22151

Page Intentionally Left Blank

PREFACE

This report was prepared under Contract NAS 1-10692, "Study to Develop a Computer Program for the Synthesis and Optimization of Reusable Launch Vehicles." The study was carried out in the period from March, 1971, to June, 1972. The study was funded by the National Aeronautics and Space Administration, Langley Research Center, and sponsored jointly by the Space System Division and the Flight Dynamics and Control Division. The United States Air Force Flight Dynamics Laboratory concurrently sponsored a companion study entitled "Analysis Program for Rapid Mission Performance Analysis of Military Flight Vehicles" which uses the executive system presented in this report.

The two studies resulted in a new, large-scale programming technique called ODIN, for Optimal Design Integration. The use of ODIN involves the linking of independent computer programs and inter-communication of common information among the programs. This report describes the executive computer program, DIALOG which implements the ODIN concept. DIALOG controls the sequence of execution of the independent ODIN programs and performs the data management function for the program inter-communication data.

TABLE OF CONTENTS

1.0 SUMMARY

An executive computing system called DIALOG has been developed for linking independent applications computer programs to form an interdependent system of programs for synthesizing engineering processes. The DIALOG executive system represents a significant departure from the usual means of forming synthesis programs as illustrated in Figure 1-1. Todays typical synthesis program is a collection of analysis programs merged together into a single computer program. The data management function is programmed into the synthesis program. DIALOG controls the sequence of execution of the independent program elements and performs the data management function by maintaining a data base of information. The data base is the common information link among the program elements.

The DIALOG executive program provides a very large scale programming system with the potential benefits listed in Figure 1-2. The basic elements of the DIALOG executive system are:

1. A library of independent applications programs.

2. A data base of control card sequences for the execution of the independent programs.

3. A language for controlling the execution of a sequence of independent programs by simple commands.

4. A dynamically constructed data base containing all interprogram data in an unstructured name oriented format. These data can be randomly selected by name at any point in the simulation.

5. A language for automatically retrieving data base information as input to any of the applications programs. An advanced information access and retrieval system is included as an integral part of the DIALOG executive system.

6. A simple technique for allowing any program in the synthesis to update the data base. The technique does not influence the stand alone operation of the program.

7. A data base of input information for each library program structured to meet the needs of the particular program. These input files may or may not contain data base interface information.

All elements of the program intercommunication are directly controlled by DIALOG. The significant advantage to the system is the rapid response to everchanging synthesis requirements. The analyst has the choice of model complexity through replacement or addition of functional program elements. The developer of new program elements is unconstrained by the requirements of the executive system. New programs may be rapidly incorporated into the program library.

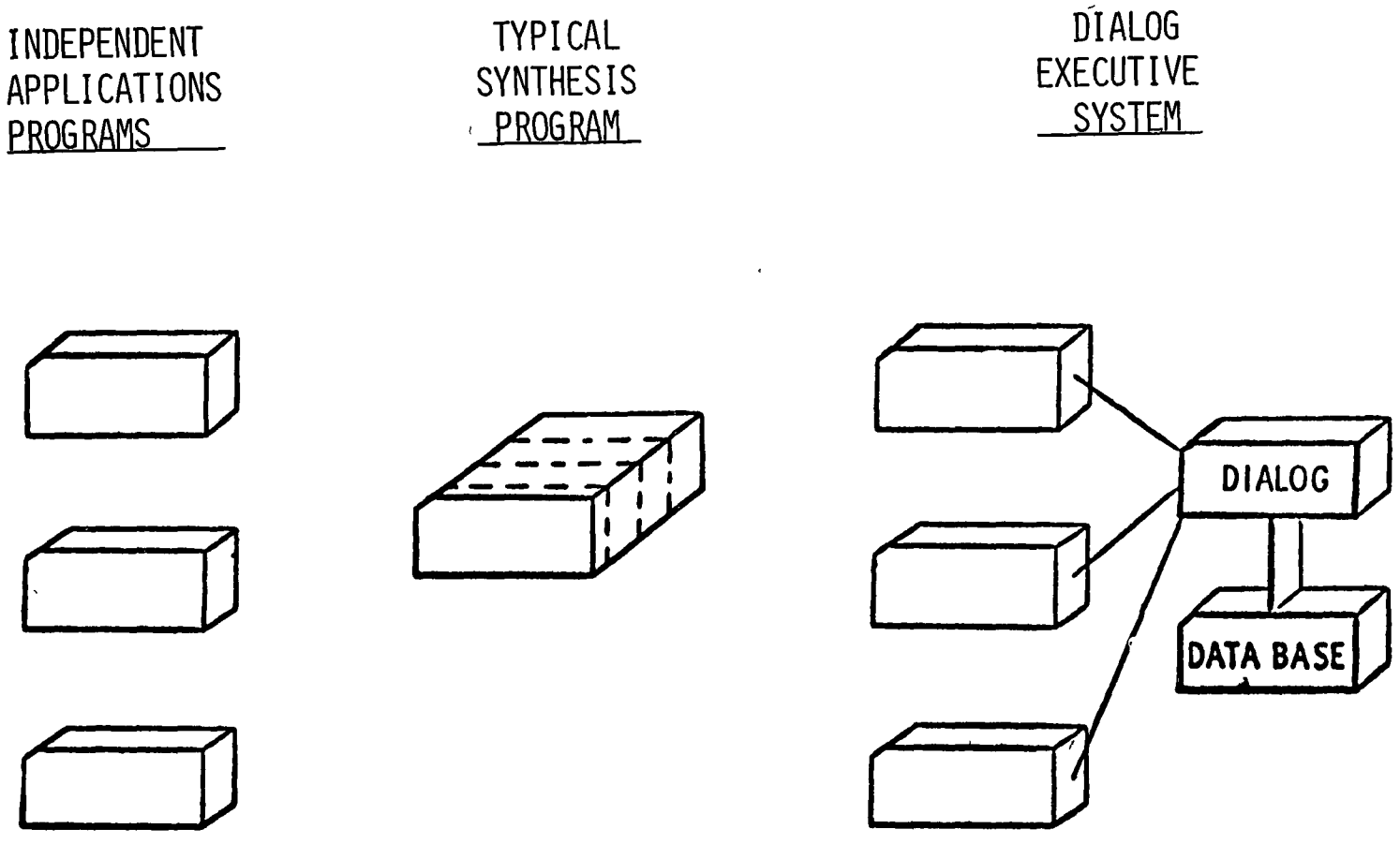


FIGURE 1-1 LARGE SCALE PROGRAMMING CONCEPTS

TAKES FULL ADVANTAGE OF PAST EFFORTS IN COMPUTER PROGRAM DEVELOPMENT

AUTOMATES MORE OF THE DESIGN PROCESS

REDUCES MANUAL DATA HANDLING BY AT LEAST AN ORDER OF MAGNITUDE

ACCELERATES THE SYNTHESIS OF LARGE SCALE SYSTEMS

FIGURE 1-2 POTENTIAL BENEFITS FROM THE DIALOG EXECUTIVE SYSTEM

2.0 INTRODUCTION

The design of an aerospace vehicle demands the involvement of specialists from all engineering disciplines. Many iterations are usually required before a suitable vehicle design emerges. The design iterations usually require from one to three months depending on the level of detail employed. Each discipline involved in the design process generally is constrained by the requirements of other disciplines, and much laborious data communication is required at each step. The interface among disciplines is often ill-defined leading to untimely or inaccurate information transfer. Under these circumstances, decisions affecting the usefulness of the end product can be based on poor or unreliable information.

The above factors have lead to increased use of the high speed digital computer to expedite the design process and improve the quality of the design information. Automation of the individual disciplines has played an increasing role in the design process for more than a decade. Structural analysis and system performance have led the way in large scale computer applications, although nearly every aspect of the design process has been automated to some degree. More recently, the merging of the technologies into a single preliminary design tool has been attempted. One successful preliminary design tool is exemplified by References 1 and 2. Here a complete synthesis of the design and mission analysis is contained in a single computer program.

The confidence gained in early simulation attempts has led to the development of more detailed and complex modules. References 3 through 8 are examples of recently developed simulation tools. However, most modern day integrated design programs tend to suffer from one or more of the following deficiencies:

a. Lack of depth in the analysis techniques.

b. Insufficient or inflexible data intercommunication.

c. Poor response time to rapidly changing design requirements.

d. Excessive computer core requirements.

By-and-large the technical depth is available in independent technology programs. The pattern of development of these programs has been the generalized multiple option approach suitable to the analysis of many classes of vehicles, each class being represented by input data. The problem arises in combining the technology programs into a design synthesis program. Computer core limitations require that resulting synthesis programs be generally more limited in scope than the individual program. As a result, the synthesis programs tend to become obsolete very quickly as the design process evolves. Very often the obsolescence occurs before any effective use can be made of the synthesis program.

The deficiencies described above have led to the development of a new design synthesis procedure called ODIN (Optimal Design Integration) described in Reference 9. The ODIN procedure shown schemetically in Figure 2-1 is a very large scale synthesis procedure which allows the selective use of existing computer programs as elements of a larger more comprehensive design simulation. Reference 9 exemplifies the technology modules which have been used with the ODIN procedure. All the depth of analysis in each technological area is maintained and the computer core requirement is no larger than the largest program element selected.

The linking of the independent program elements is controlled by the executive computer program, DIALOG which also controls the communication of information among the independent program elements. An input language to the DIALOG executive system provides the user with the ability to formulate the design problem at the task level in much the same manner as is currently employed in the design process. As much or as little of the design process may be automated as suites the particular application. The design staff directly controls the specific information being communicated from program-to-program and from the design simulation to the design staff.

Since the system uses existing checked out computer codes as building blocks in performing the design tasks, no program development is usually required. The program elements are usually in common use throughout the design staff and therefore readily usable in the design simulation. No more effort is required to establish an automated design sequence than that required to establish a single design cycle by ordinary means. The same computer codes are generally used in either case. Once established the automated procedure can be used many times for design perturbations, and can be quickly changed to suit changing design requirements. The designer never relinguishes his option to perform any task by some alternate means including hand calculation.

The current documentation describes the control and communication language of the DIALOG executive system. It will become apparent that the DIALOG executive system is not specifically tied to design simulation. Although originally developed to implement the ODIN procedure, the DIALOG executive system is generally applicable to any engineering process. Little reference is made to design simulation in this report. The library programs which DIALOG controls are referred to as applications programs to avoid the implication that only technology oriented tasks may be employed.

Indeed there are many "utility programs" used in ODIN which perform non-engineering tasks but are quite useful in any process.

Section 3 contains a general description of the DIALOG executive functions. Much of the programming detail has been ommitted providing an overview of the system capability. Section 4 is a detailed description of the installation of the DIALOG executive system at the NASA Langley Research Center Computer Complex. The nature of the DIALOG executive system requires

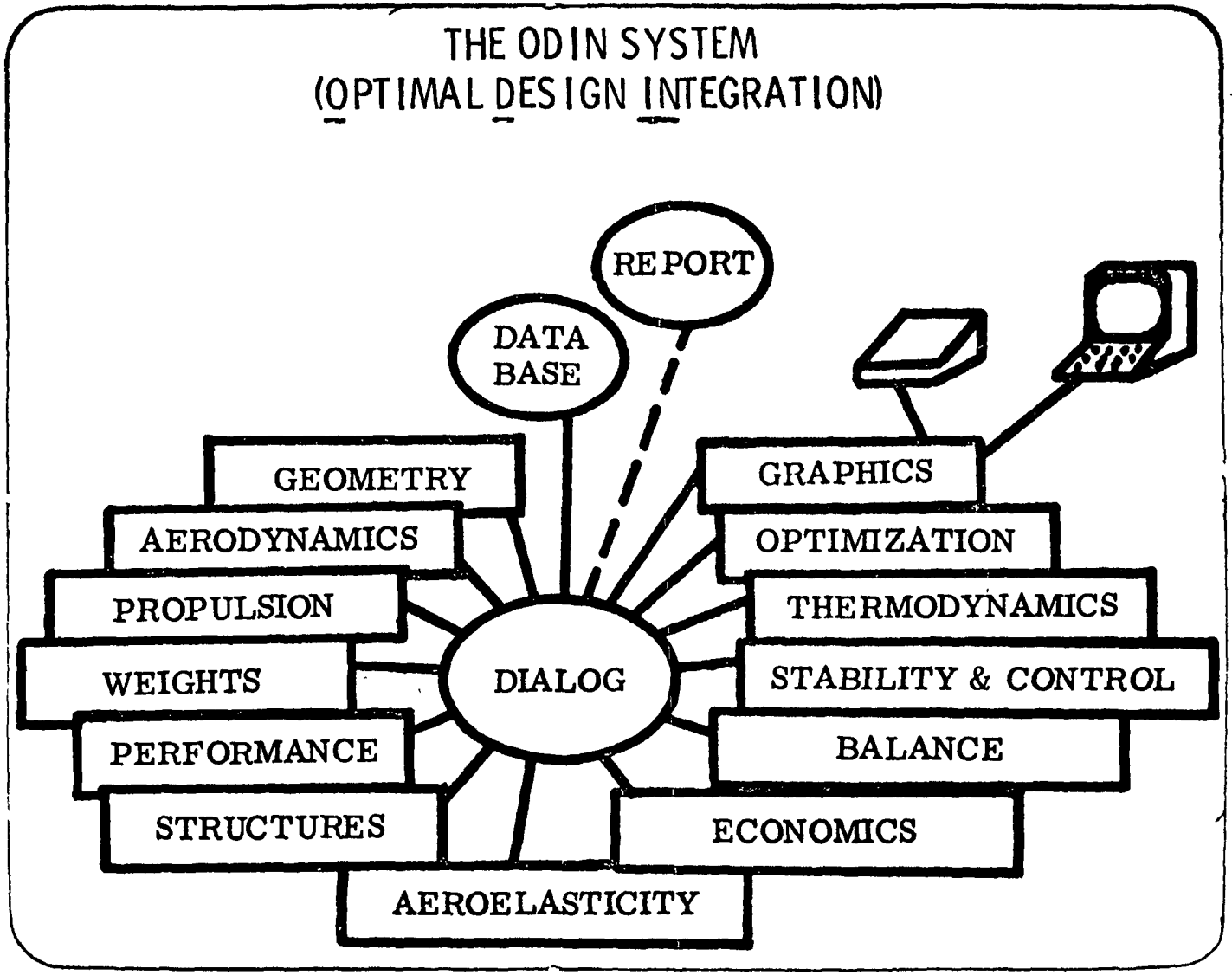


FIGURE 2-1 CONTRIBUTION OF DIALOG TO THE ODIN PROCEDURE

a close relationship with the operating system on which it is installed. Although representative of the installation on other CDC 6600 systems, the LRC system installation is presented as an example. Section 5 is devoted to the use of the DIALOG languages developed for the purpose of linking independent programs and communicating information among them.

3.0 DIALOG FUNCTIONS

Usually the submission of a computational sequence to the digital computer involves the execution of a single computer program with possible repetitive evaluation of successive data cases. When using the DIALOG executive system, submission of a computation may involve the sequential execution of many programs to obtain a complete analysis. For example the repetitive execution of program sequences will be required for parametric studies or optimization problems. The use of the DIALOG executive system also affords the analyst the opportunity to conveniently communicate data from stratified sources among the programs in the execution sequence. A discussion of these two basic functions is presented in the following paragraphs.

3.1 Computer Control Card Assembly

On a digital computer the execution of a single program is governed by a set of control cards which provides instructions to the computer system for compiling and/or loading the specified program. The control cards are peculiar to each computer system and installation. The control cards rarely employ user oriented format. For example, Figure 3-1 presents typical control cards for an elementary FORTRAN compilation and execution of the same program on a CDC 6000 series computer, an IBM 360 series computer and a UNIVAC 1108. Further, control cards on any computer of a given series or manufacturer can vary from installation to installation. Figure 3-2 shows the control cards to retrieve from storage and execute a machine language program at three different installations. Though each installation uses a CDC 6600 computer, the differences in compilers, loaders and peripheral hardware result in entirely different control cards to perform the same functions.

3.1.1 Execution of an Applications Program

In actuality, to retrieve and execute an applications program, several independent programs must be executed. Collectively the control cards required to execute an applications program may be referred to by name such as PGMA or PGMB. These independent program executions which we call "control cards" are all part of the computer operating system. System programs of the type called by control cards bear a similar relationship to the computer operating system as do independent applications programs PGMA and PGMB to the DIALOG executive system, Figure 3-3. This analyogy may be formalized as follows:

> "The operating system employs independent system utility programs to retrieve, compile and execute a given applications program. The DIALOG executive system employs control card sets to synthesize an engineering process."

DIALOG contains a higher order programming language which carries out the analysis function by linking control card sets rather than carrying out the individual control card functions.

CDC 6000 SERIES COMPUTING SYSTEM

```
RFL,60000.
FTN,OPT=0.
LGO.
7-8-9
       SOURCE DECK
7-8-9
       DATA DECK
6-7-8-9
```

IBM 360/67 SERIES COMPUTING SYSTEM

```
//EXEC FORTGCG
//FORT.SYSIN DD *
       SOURCE DECK
/*
//GO.SYSIN DD *
       DATA DECK
/*
```

UNIVAC 1108 SERIES COMPUTING SYSTEM

```
@ FR5 MAIN
       SOURCE DECK
@ XQT MAIN
       DATA DECK
@FIN
```

FIGURE 3-1 TYPICAL CONTROL CARDS TO COMPILE AND EXECUTE A FORTRAN PROGRAM

NASA LANGLEY RESEARCH CENTER

```
FETCH,A0000,SPRA00,BINARY,,OVL.
OVL.
7-8-9
      {DATA DECK}
6-7-8-9
```

A. F. FLIGHT DYNAMICS LABORATORY

```
ATTACH,OVL,A0000,CY=1,MR=1.
OVL.
7-8-9
      {DATA DECK}
6-7-8-9
```

NASA AMES RESEARCH CENTER

```
LIBCOPY,ALIB,OVL/BR,A0000.
LODE,I=OVL,O=UNSATED.
7-8-9
      {DATA DECK}
6-7-8-9
```

FIGURE 3-2 EXAMPLE CONTROL CARDS TO RETRIEVE,AND EXECUTE A MACHINE LANGUAGE PROGRAM AT THREE CDC 6600 COMPUTER INSTALLATIONS

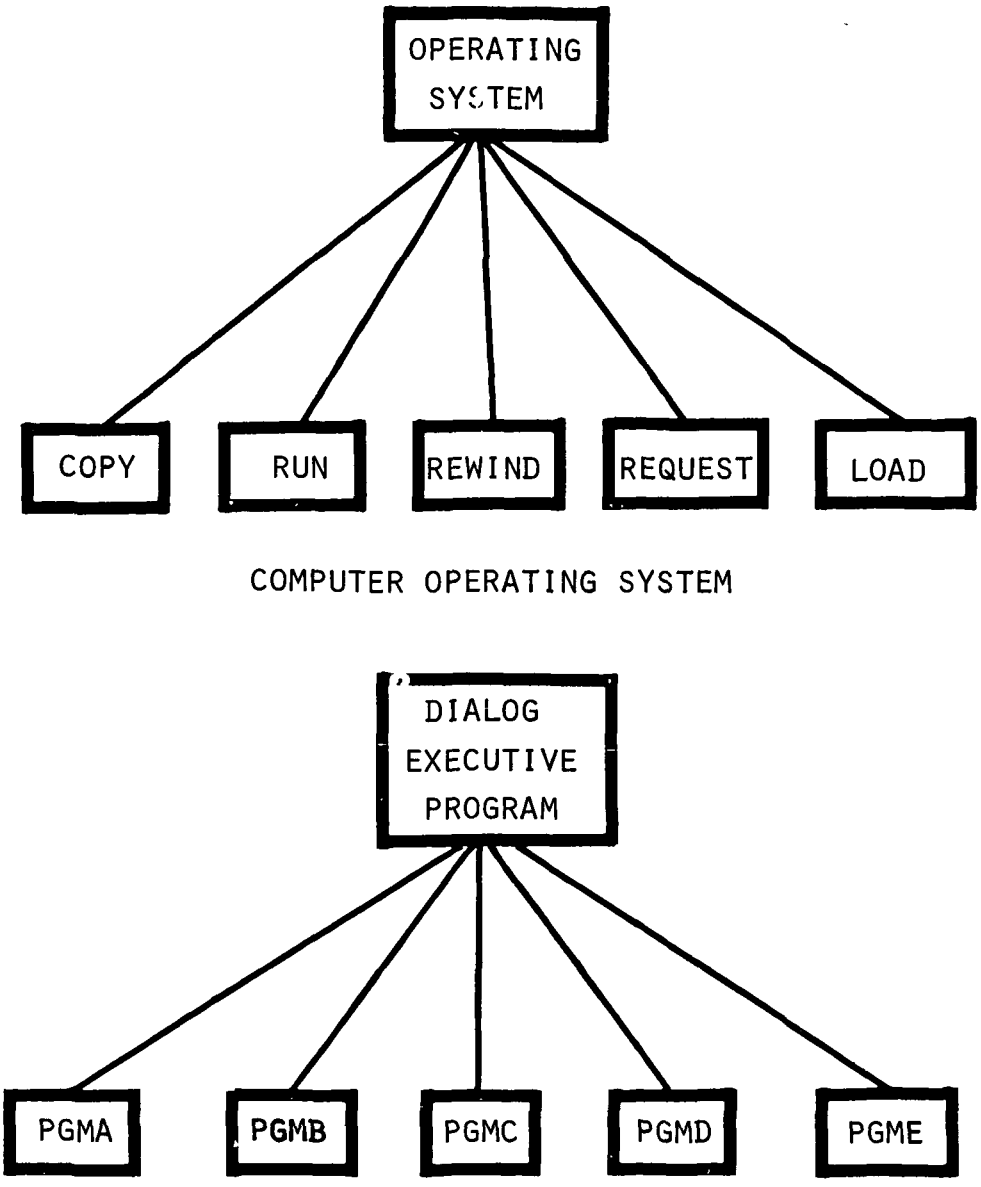


FIGURE 3-3 ANALOGY BETWEEN OPERATING SYSTEM AND DIALOG EXECUTIVE SYSTEM

3.1.2 Creation of a Control Card Data Base (CCDATA)

The nature of the computer operating systems with regard to the execution of applications programs via a sequence of control cards has led to the development of the control card data base concept. The data base contains all the control card sequences required to retrieve and execute a library of applications programs. Collectively, any sequence of control cards necessary to execute a given applications program is referred to by a name.

```
EXECUTE PGMA
```

The name, PGMA, is assigned when the control card sequence is stored in CCDATA. In the remainder of this section details of the control cards will be omitted. The control card sequences will be referred to by the name under which it is stored. The command to execute the control card sequence:

```
EXECUTE PGMA
```

will be referred to as a control directive (i.e. the EXECUTE directive). Other control directives will be described, each performing a "control function" in the execution sequence. Collectively the control directives form the DIALOG control directive language.

The control directive language is input to the DIALOG executive program. DIALOG processes all input to all programs and performs certain functions based upon the control directives encountered. To distinguish DIALOG input from the input of applications programs, the control directives must be delimited as follows:

```
'EXECUTE PGMA'
```

The creation of the control card data base is a function of DIALOG. It reads from input cards the control card sequences to retrieve and execute programs from the library. The control directive which creates the control card data base is:

```
'CREATE CCDATA'
```

Following this directive, the control card sequence such as those illustrated in Figure 3-1 and 3-2 are entered into CCDATA by name:

```
PGMA =

control
 card
sequence
```

Any number of these control card sequences may be entered into CCDATA,

each representing the retrieval and execution of an applications program. Once established, CCDATA may be accessed by the DIALOG executive through the EXECUTE directive described above.

3.1.2.1 Storage of the Control Card Data Base

The creation of a control card data base does not insure the availability of the data base for a future use. It is only available for the run in which it was created. However, it can be saved by the execution of a special utility procedure. The special procedure is simply another control card sequence analogous to an applications program control card sequence. Usually the procedure is itself stored in CCDATA and executed by the EXECUTE control directive as follows:

```
'EXECUTE CCSAVE'
```

The CCSAVE procedure is one of a series of utility functions which have been developed for the DIALOG executive system. The utility functions include plotting, picture drawing, abnormal end and file printing procedures.

The CCSAVE save procedure is installation dependent and hardware dependent (i.e. tape, disk or data cell). As such it is generally provided in the basic control card data base created at the time the DIALOG executive system is installed. Other utility procedures which will be described later in this report are also provided in the basic system.

3.1.2.2 Updating the Control Card Data Base

Once established, the control card data base can be freely accessed by the EXECUTE control directive without regard to the actual control card sequence involved. Usually the control card sequences do not change. However the DIALOG control directive language permits the complete replacement of existing control cards sequences or the modification of individual cards. The UPDATE control directive is used for this purpose. UPDATE is described in detail in Section 5.

3.1.3 Execution of a Sequence of Applications Programs through Control Card Linkage

Now consider the problem of sequential execution of more than one application program using the DIALOG executive system. Assume the following three control directives:

```
'EXECUTE PGMA '

'EXECUTE PGMB '

'EXECUTE PGMC '
```

The function of DIALOG is simply to retrieve the control cards sequences for PGMA PGMB and PGMC from the control card data base, CCDATA and queue

them sequentially on a file called CONTROL. The CONTROL file is then interrogated by the operating system which performs the various control card functions. Included in these control card functions are the executions of the desired applications programs. Figure 3-4 illustrates the relationship between the DIALOG executive and the operating system. Progressing from left to right in Figure 3-4, the control directives to execute control card sequences are read by DIALOG, which "builds" the control card sequences from information stored in the control card data base and passes the control card sequences to the operating system.

The physical link among the control card sequences of the various applications programs and DIALOG is an operating system utility such as the CCLINK utility on the CDC 6600. CCLINK is executed by a control card which forces the operating system to read control cards from an alternate file built by DIALOG. In the DIALOG executive system CCLINK is used to link the execution sequences of an applications program to DIALOG, then from DIALOG to the next applications program, then to DIALOG, etc. This is illustrated in Figure 3-5. So the DIALOG executive program first constructs the control card sequences to execute applications programs then through the use of CCLINK, provides for the re-execution of DIALOG to process the application program input and output data. Details of the operation of the example utility CCLINK are contained in Appendix A. Similar utility programs are usually available in most third generation systems.

3.1.4 Repetition of Control Card Sequences

Thus far we have described a capability which permits the execution of control card sequences in an arbitrary manner using higher order readily understood commands. This is achieved by the creation of a control directive language which replaces the control card sequences such as those in Figures 3-1 and 3-2. However, there are two additional capabilities which exist in the DIALOG executive system which are not possible simply by queuing control card sequences. These include the conditional branching logic described below and the maintenance of a design data base described in Section 3.2.

The DIALOG executive system permits automatic repetition of control card sequences by a system of conditional branching logic. This capability is achieved by extension of the control card directive language in the following manner:

```
'DESIGN POINT1'

'LOOP TO POINT1'

'IF V1 .LT. V2'
```

The DESIGN directive establishes an identifier in the execution sequence where control may be returned (or skipped to). The LOOP TO directive points to the identifier to which control is to be returned. The IF

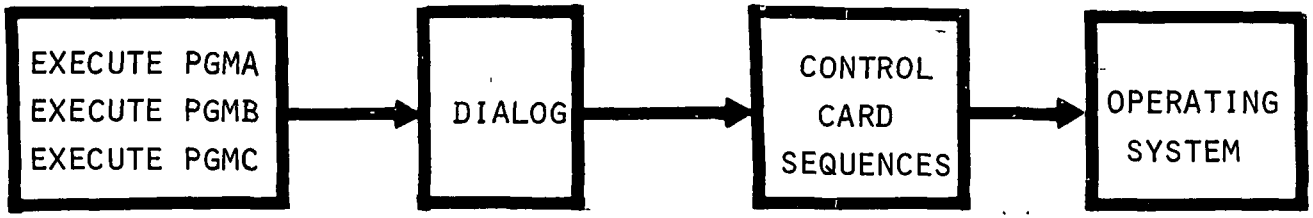


FIGURE 3-4 ILLUSTRATION OF THE RELATIONSHIP BETWEEN THE DIALOG EXECUTIVE AND THE OPERATING SYSTEM

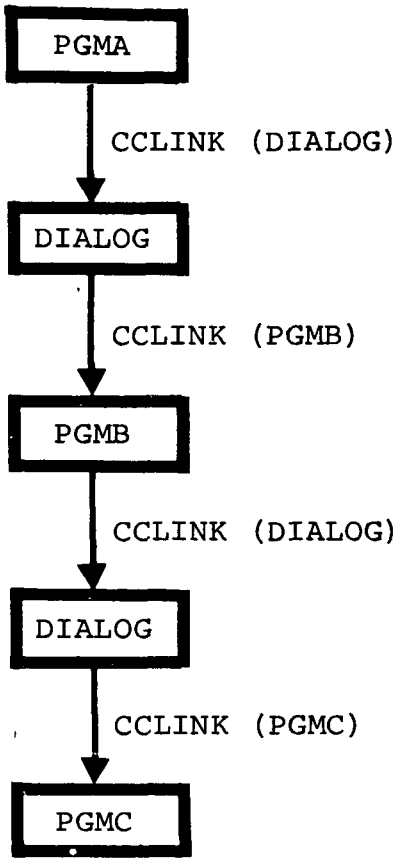


FIGURE 3-5 ILLUSTRATION OF THE FUNCTION OF THE SYSTEM UTILITY, CCLINK

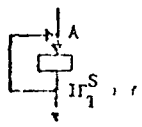
directive is a conditional operator based upon design dependent logic. If absent, the LOOP TO directive is a mandatory branching command. V1 and V2 are example values which may be constant or computed in any of the applications programs. In the latter case such values must have variable names and be defined in the design data base. A description of the design data base is given in Section 3.2.

In general the DIALOG executive system permits a complicated system of analysis loops for satisfying a variety of matching constraints. It is not possible or necessarily desirable to rigidly define the topology of the system of computational loops. Instead, the analysis sequence to be performed is defined within the control directive language. This technique allows the analyst complete freedom in specifying the computational sequence; no limit is placed on the complexity of the analysis.

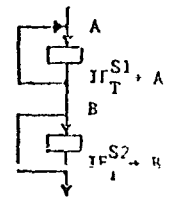
Any number of loops can be created using the LOOP TO and conditional IF control directives and the associated DESIGN control directive. Using the symbolic notation:

$$IF_T^S \longrightarrow A$$

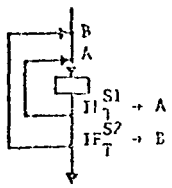
to indicate if the statement S is true go to A, it is apparent that series loops, nested iterative loops and combined series and nested loops can be constructed. For example:



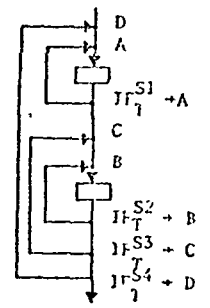
a. SINGLE LOOP



b. TWO SERIES LOOPS



c. TWO NESTED LOOPS



d. TWO SERIES LOOPS AND TWO NESTED LOOPS

The IF tests employed encompass the standard set of six tests in FORTRAN; although the form of the DIALOG control directive language test differs in form to that of FORTRAN. The six tests are:

'IF V1 .LT. V2'	IF (V1 < V2)
'IF V1 .GT. V2'	IF (V1 > V2)
'IF V1 .LE. V2'	IF (V1 ≤ V2)
'IF V1 .GE. V2'	IF (V1 ≥ V2)
'IF V1 .EQ. V2'	IF (V1 = V2)
'IF V1 .NE. V2'	IF (V1 ≠ V2)

As noted previously V1 and V2 are constants or variables constructed in the design data base or constructed within any independent program in the synthesis and passed to the design data base.

The ability to select alternative program execution paths based on design dependent logic is illustrated below:

```
.........

'EXECUTE PGMA'

'DESIGN POINTA'

'EXECUTE PGMB'

'LOOP TO POINTB'

'IF V1 .EQ. V2'

'IF V3 .LT. V4'

'EXECUTE PGMC'

'LOOP TO POINTA'

'DESIGN POINTB'

'EXECUTE PGMD'
```

The above control directive sequence defines the execution of PGMA and PGMB with a conditional loop (in this case, skip) to POINTB. If neither of the IF conditions is satisfied, PGMC is executed followed by a mandatory loop back to POINTA.

In general, both V1 and V2 may be defined by the analyst or alternately either may be a variable computed by any of the application programs. In the latter case such variables must be defined in the data base as described in Section 3.2.

3.2 Data Management Function

The usual manner of transferring information from one program to another is by use of a structured file. This simply means program A is coded to create a file of data in exactly the same format as required by program B. In the sequential execution of the two programs, the structured file created by A is passed to B via parameters on the execution control card.

In the DIALOG executive system the above means of transfer of information is possible and often employed. However, in the communications of information from one program to another, it is not always possible or even desirable to create a structured file of input data for each program. Often only a few stratified bits of common information are required for each program. Usually a different set of data and a different order is required for each program. The DIALOG executive system maintains a name-oriented data base containing an unstructured set of data accessible by all programs. The data base file of information is dynamically constructed by DIALOG as the analysis proceeds. The file is resident on disc or tape at the users option. Construction of the design data base involves the following tasks:

a. Search to see if the variable name exists within the fixed field of the data base.

b. If not, locate a vacant location in the data base and install the information and the name.

c. Otherwise, replace the information associated with the name.

Additional information can be added or existing information may be updated by the analyst at any point in the analysis. Data base limits are discussed in Section 4.

The design data base can be updated by any of the applications programs as the analysis proceeds. Updating the data base by the applications programs involves tasks similar to those described above.

Further, the analyst may specify that all information created by the applications programs shall be placed in the design data base. In this case the tasks are identical to that of creating the design data base. Figure 3-6 illustrates the data interplay among the applications programs. Consider a variable stored by the name, WEXPAR. Assume WEXPAR is computed in program A (PGMA) and subsequently used by program B (PGMB). Schematically this is illustrated in Figure 3-6. Any number of subsequent programs may access WEXPAR or alternately update the value of this variable. All interface between the applications

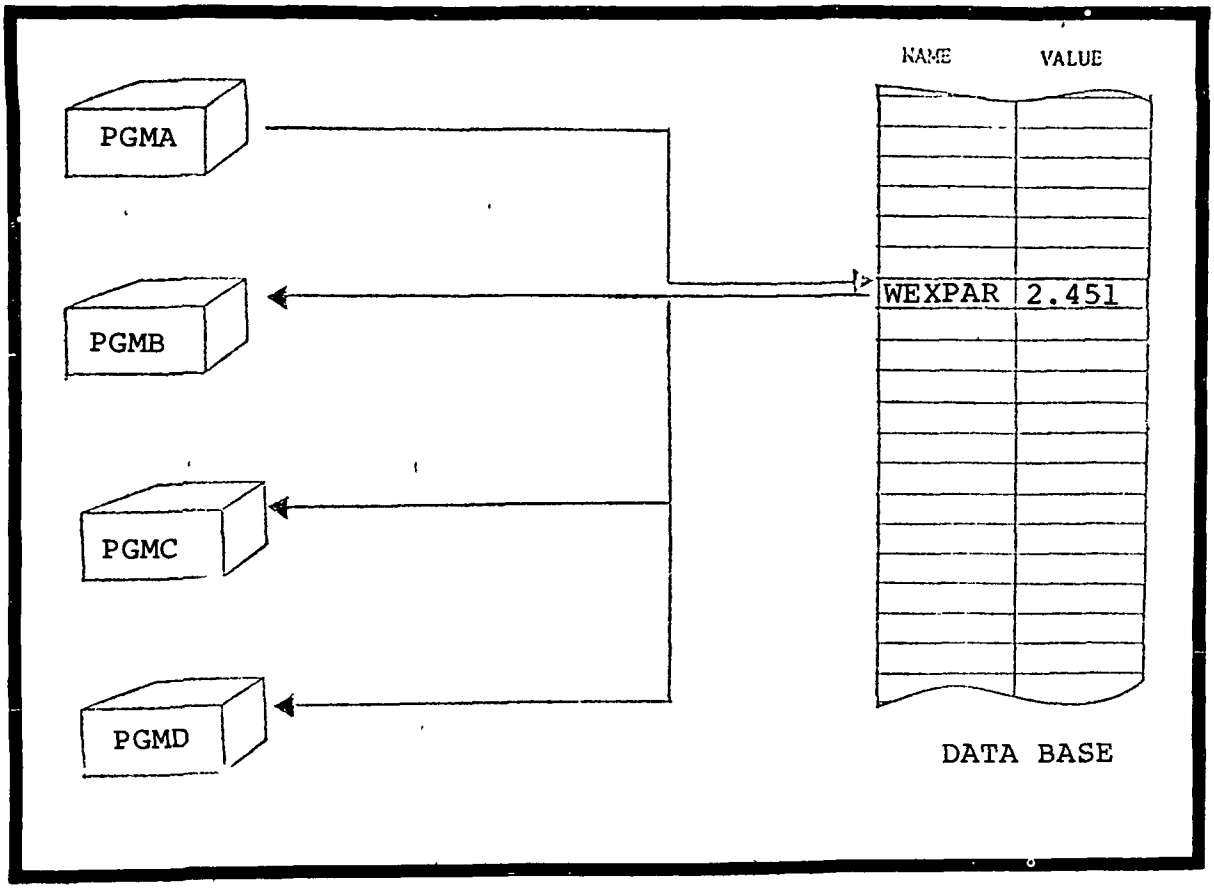


FIGURE 3-6 DATA INTERPLAY AMONG APPLICATIONS PROGRAMS

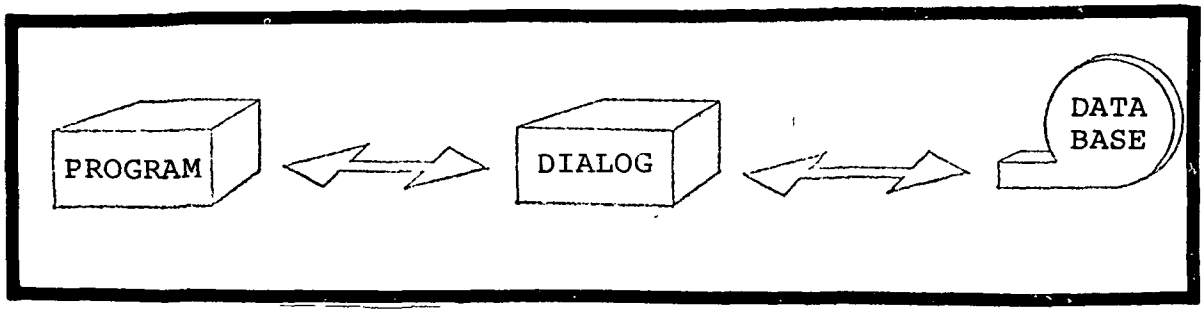


FIGURE 3-7 DIALOG EXECUTIVE CONTROLS ACCESS TO DATA BASE

programs and the design data base are performed by the DIALOG executive as illustrated in Figure 3-7. The same is true of the interface between the analyst and the data base. All data requests are addressed to the DIALOG executive.

Data base information may be accessed by the analyst for placement into the input stream of any of the applications programs. The DIALOG tasks in performing this function are:

a. Search to see if the variable name encountered exists in the data base.

b. If not, ignore the access request.

c. Otherwise, retrieve the information associated with the access request.

d. Replace the variable name encountered with the data base information.

The technique employed in the storage and retrieval of data base information is discussed in the following paragraphs. They involve the extension of the control card directive language described in Section 3.1 as well as the creation of a new intercommunication language for passing design information from one applications program to another.

The new language contains a simple set of instructions which will be referred to as communication commands. Communication commands are physically inserted into the applications programs input data. In general, these commands are either removed or replaced by the DIALOG executive program before the input data is processed by the applications program.

The communication commands form the basis by which unstructured information is passed from one applications program to another. The communication commands are delimited in exactly the same manner as the control directives. Complete syntax rules for the language are given in Section 4.

3.2.1 Data Base Information Transfer System

Data base information transfer is accomplished through a rapid search by name. Search speed is obtained by the use of "hash" and "collision" methods of Reference 10. This approach is more efficient than the more usual linear sequential search which starts with the first name in the table and proceeds sequentially until the desired name is located and the corresponding value is retrieved.

The hash and collision data transfer system operates in the following idealized manner:

a. Take the variable name and treat the binary representation of this word as an integer;

b. Find the remainder when the integer representation is divided by the

number of elements in the data base. This is equivalent to the FORTRAN MOD function which is a very rapid machine operation.

c. Use the remainder as the nominal location or "hash" location of the variable within the data base. This assures the location derived will fall within the data base limits.

d. Check to see if the location is in use since more than one variable name may hash to this location. If this location has already been used for another variable name store the new variable in the next vacant location and provide a pointer to this location in the data base entry originally searched. This pointer is called a "collision" pointer. Each entry has associated with it a name, a value, a hash address and a collision pointer.

e. The retrieval process operates in the same manner. The name is converted to nominal retrieval location. If that location contains the wrong name, the specified alternate location is searched for the desired name, etc. until the desired name is found and the variable value is retrieved.

Figure 3-8 illustrates the hash and collision method. Suppose the binary representation of three variables A, B and C are identical to a fourth variable stored in the data base. Upon initial entry, the name A "hashes" to the occupied location. After unsuccessful comparison with the existing name at that entry, a new location for A is defined and a collision pointer is stored at the original entry forming a link to the new location. Once a location is established for A, the information (value) is stored or retrieved. The name B is also "hashed" to the original location. An unsuccessful comparison with the existing entry sends B to the location where A is stored via the pointer described above. An unsuccessful comparison with A causes the next available location to be defined for B. A pointer to the newly defined location is stored at the entry for A forming the link to B. This chaining process described can be continued to the limits of the table.

In numerical experiments with a 2000 word data base filled approximately 75 per cent, it was found that the average name can be retrieved in less than two attempts (fetches). This would compare with 750 fetches using a linear search for information retrieval. In practice using DIALOG, approximately 9000 values per second may be retrieved on the CDC 6600. This figure varies but is less affected by the amount of data stored than by the internal numerical pattern produced by the variable names stored. It is difficult to control numerical uniqueness since the data base names are arbitrarily chosen by the user. However, in all past experiences with DIALOG collisions have never exceeded 20 per cent. The access time is essentially unchanged with data base size, while the linear search technique increases in access time in proportion to data base size.

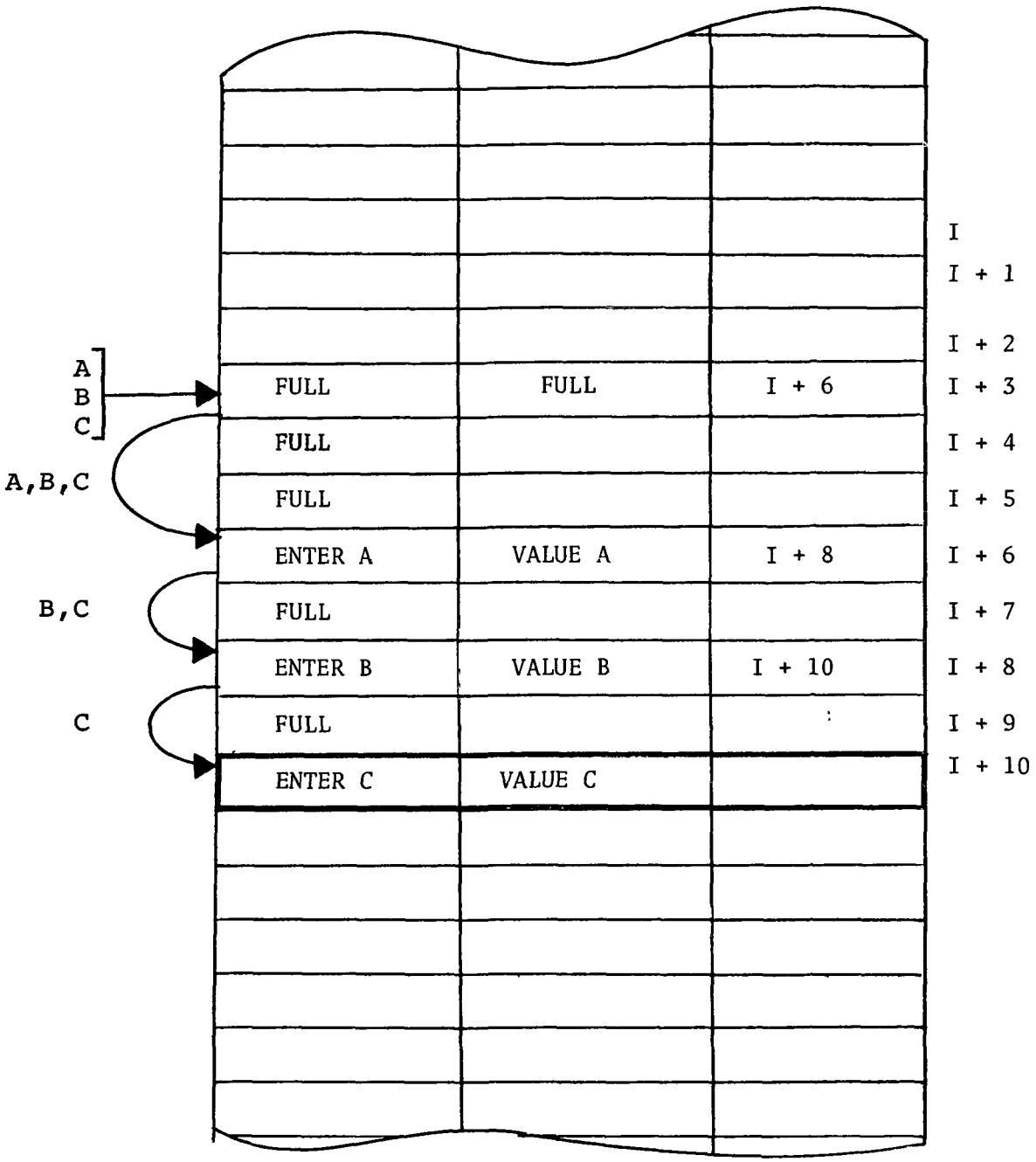


FIGURE 3-8 IDEALIZED INFORMATION RETRIEVAL SYSTEM

3.2.2 Creation of a Design Data Base

The design data base, DBASE, is created in much the same manner as the control card data base. The two data bases are similar in construction and occupy the same computer core locations but at different times. The data bases consist of two distinct parts, a free storage array of packed information and a directory of names and pointers to the actual data in the free storage array. Space in the free storage array is allocated as required by the user and/or the applications programs. As the information is stored in the free storage array, the directory is constructed as described in Section 3.2.1. Access to the data base is always through the directory. Both the directory and the data base have certain attributes which distinguish them from one another. Among these are:

- Total number of data base entries
- Number of computer words per data base entry
- Total number of directory entries
- Number of words of descriptive information associated with each name

For example, CCDATA elements are 8 computer words (8 words = one card) in length since CCDATA is used for storing control card sequences. DBASE elements are two computer words in length since DBASE is used for storing design type data in BCD format.

The construction techniques employed in creation of the design data base as well as the control card data base are easily extendable to a multiple data base involving many combinations of attributes such as data type and technology origin.

The design data base, DBASE is created with the control directive:

'CREATE DBASE'

This directive is followed by a file of information containing the necessary communication commands to initially establish the data base. These communication commands are described below. It is not essential that any information be initially placed in the data base. The dynamic nature of data base maintenance permits information to be added at any point in the execution sequence.

3.2.2.1 Adding Information to the Design Data Base

The basic communication command available to the analyst is the ADD command. It permits a variable name and value or values to be placed in the data base:

'ADD name = value, value,'

Any number of values may be added for a given name. The number of values associated with the name is also the number of locations reserved in the data base for that information. Later modifications to the information can not create more data base space. The values may be real, integer, hollerith or logical. The data type is immaterial since the information is stored in coded or character format.

A single ADD command may be used for creating or updating many information sets.

```
'ADD V1 = 25., V2 = 30, V3 = ALPHA, V4 = .TRUE.,

      A = 10., 15., 20., 25., I = 4, 5, 6'
```

The data type is specified by the input. Any of the four common types of variables may be entered into the data base. The format of the ADD statement is patterned after the FORTRAN NAMELIST feature and indeed has the same characteristics and utilization rules. For example, all name/value sets are separated by commas (,); all elemental values of an array are separated by commas; the entire statement (command) is delimited. In the case of NAMELIST, the delimiter is a dollar ($) sign; in the case of ADD command the delimiter is (').

However, the ADD command has additional capability not present in the FORTRAN NAMELIST feature. The value associated with the ADD name may be a previously defined data base variable name:

```
ADD V1 = V2,
```

The effect of the above command is to transfer the information associated with V2 to the data base space assigned to V1. V1 may or may not exist prior to the ADD command. If V1 did not exist, space will be created in the data base as the information is transferred. If V1 did exist, then the information in V1 will be replaced by the information in V2. The transfer of information from one data base location to another is generally limited to scalar quantities. Complete rules are given in Section 5. Finally the ADD command may be used for transferring multiple constants in the data base:

```
'ADD V1 = 5 * 0.,'
```

In the above illustration, V1 will be a data base array name. Five zero values will be stored.

3.2.2.2 Combining Data Base Information

The ADD command capability thus far described includes the addition or modification of data base information with either constant or variable type information. The ADD command may also be used for combining existing data base information with other data information or constant information:

```
'ADD V1 = V2 * K,'
```

or

```
'ADD V1 = V2 * V3,'
```

In the above statements V1, V2 and V3 are data base variables and K is a constant. V1 may be a new or existing data base variable. The operation illustrated above indicates a multiplication of the two numbers on the right side of the equal (=) sign prior to transferring the resulting information to the space allocated to V1. Any algebraic operator may be employed as follows:

```
+ addition
- substraction
* multiplication
/ division
** exponentiation
```

More than one operation may be performed on the right side of the equal (=) sign.

```
'ADD V1 = V2 + V3 * K,'
```

Up to ten operations may be performed within a single ADD command. However, the hierarchy or order of the operations is not the same as FORTRAN. For example, in the above illustration, V3 is added to V2, then the sum is multiplied by K. This is a significant departure from the hierarchy employed in FORTRAN. The basic rule in combining variables with the ADD command is:

> "The operations are performed in a serial manner analogous to that employed in a hand calculator."

The operations start from the equal sign and progress to the right. The first variable is combined with the second. The result of that operation is combined with the third. The result of that operation is combined with the fourth, etc. It is very important from the outset that the analyst understand this principle.

In summary, the ADD command gives the analyst the ability to add, modify and combine information in the data base at any point in the execution sequence. The only constraint is that its occurrence must be within an input data set for an applications program or as a result of a CREATE or UPDATE control directive for DBASE. Figure 3-9 illustrates the possible locations for ADD commands. ADD commands can not be mixed with control directives. It may be noted that the ADD command serves as an instruction only to the DIALOG executive and is not a part of the normal input data to the applications program. As such, the ADD command is removed from the input stream as it is processed.

The combining of variables is very useful when coupling computer programs from independent sources. One example is the matter of units conversion.

(a) Adding data to a new data base

```
CREATE DBASE
.............
.............
ADD V1 = 25., V2 = 30.,
.............
.............
.............
VN = 60.,
.............
(end of file)
```

(b) Adding data to an existing data base

```
UPDATE DBASE
.............
.............
.............
ADD V1 = 25,
.............
(end of file)
```

(c) Adding data during the execution sequence

```
EXECUTE PGMA
.............
.............
ADD V1 = V2,
.............
(end of file)
```

FIGURE 3-9 POSSIBLE LOCATIONS FOR THE ADD COMMAND

Very often computer programs use different unit systems. When coupling such programs the output of one computer program may not provide compatible data for the input to another. The ADD command gives the analyst an immediate means of providing that essential data compatibility without modifying any applications programs. The DIALOG executive system does not exclude the possibility of automatically providing data compatibility among applications programs at a future date.

The ADD command is also useful for performing simple interface transformations. Indeed, a limited FORTRAN capability exists as part of the communication command language. However, it is not intended to replace FORTRAN or other languages. It simply augments existing analysis tools. Later discussions will show that full FORTRAN (or any other common language) capability is immediately available within the DIALOG executive system for complex data transformation problems.

3.2.2.3 Defining Variables and Reserving Space in the Data Base

It is often desirable in using the DIALOG executive system to reserve space in the data base before the information is actually generated. The DEFINE command was developed for this purpose. As with the ADD command, the DEFINE command may be employed anywhere within the execution sequence. However, it is most likely to be used in conjunction with creating or updating the data base. The format is as follows:

'DEFINE V1 = n, description,'

V1 is a new or existing data base entry. The number of locations reserved is n. If V1 is an existing variable, n is ignored. If n is absent from the command, one is assumed. The description is a short hollerith description briefly describing the variable V1. The length of the description is typically three computer words. This is not a hard limit and can be altered with the alteration of a dimension statement in DIALOG. The description is printed together with the data name and value when the control directive:

'PRINT DBASE'

is employed. Figure 3-10 is an illustration of a printed data base.

3.2.2.4 Identification of Applications Program Data

In addition to the ADD and DEFINE commands, there exists a special "command" for identifying data. The format is:

'. comment'

No action is performed as a result of this command. It is useful only as an identifier for other data. For example, consider an application program which uses formatted input (i.e. numbers with no identifiers or names associated with them). The comment command may be used to

```
* * * * * THERE ARE CURRENTLY    19 DATA BASE ENTRIES AS FOLLOWS * * * * * * * * * * * * * *

NAME        LOCATION DIMENSION  CURRENT VALUE(S)         ORIGIN   DESCRIPTION

AESOP            39        1                             DEFINE   AESOP USED IN SIMULATION
ALPHA             7        2   0.525E+06                 AESOUT   AESOP CONTROL PARAMETERS
                               0.2715E+01
BUILD            19        1   0                         DEFINE   DYNAMIC DB BUILD OPTION
CRDSKP            1        1   32                        ADD      CARD SKIP OPTION
DROUMP           31        1                             DEFINE   CAUSES DATA BASE DUMP
DISPLAY          13        1   WPAYLO                    INITAL   SCOPE DISPLAY FUNCTION
FLTIME           23        1   28.0920000                DEFINE   TOTAL ELAPSED TIME
JJBEST           25        1   1                         AESOUT   BEST PERFORMANCE EVALUATION
JJJ              17        1   0                         AESOUT   AESOP EVALUATION COUNTER
MIXR              3        1   2.7149999999999989200     ADD      PROPELLANT MIXTURE RATIO
PJNID            21        1   *ODIN       JAN 22, 72    DEFINE   IDENTIFICATION FOR DIALOG RUN
TCOSTB           35        1   48673.855937806656658     ADD      TOTAL COST OF BOOSTER
TCOSTN           37        1   0.19937115923077E+05      DAPOUT   TOTAL COST OF ORBITER
THRUST            5        1   525000.000000000000000    ADD      ENGINE VACUUM THRUST
VSTB             29        1   0.91561786551872E+04      WTOUT    BOOSTER STAGING VELOCITY
VSTGO            33        1   0.25825309779844E+05      WTOUT    ORBITER STAGING VELOCITY
WGROSB           11        1   0.26662821532874E+07      WTOUT    GROSS WEIGHT OF BOOSTER
WGROSO           15        1   0.71802039902725E+06      WTOUT    GROSS WEIGHT OF ORBITER
WPAYLO           27        1   0.31847761977751E+05      WTOUT    PAYLOAD OF THE LAUNCH SYSTEM

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Data base entry name

Data base location and dimension of information

Current data base information

Description entered with a 'DEFINE' command

The command which last updated the information the information in the data base. ODIN output namelist names are treated as commands.

FIGURE 3-10 EXAMPLE OF PRINTED DATA BASE FOR 19 ENTRIES

identify the data elements within the input data stream. The affect of processing this command through DIALOG is that the command is simply replaced with blanks. If the resulting card is entirely blank, then the card is "removed" from the input stream.

The effect of using the comment card with the execution sequence is to provide some self documentation. It serves to identify data which is not generally recognizable as it stands. The comment command is somewhat analogous to the comment card in the FORTRAN language.

3.2.3 Communicating Information from the Data Base to the Applications Programs.

The three commands described above, the ADD command, the DEFINE command and the "." command (comment), basically provide user interface with the data base. This paragraph and the next paragraph deal with applications program interface with the data base.

In this section the passing of information from the data base to the applications programs will be discussed. Section 3.2.4 deals with the passing of information from the applications programs to the data base.

3.2.3.1 Modifying Program Input to Communicate with the Data Base

Development of the DIALOG executive system is based on the premise that independent applications programs can be made to communicate with each other without significant modification through a data base. By following this premise a method of communicating data base information into each program has been devised. No modification to the program input data code is required by this method. The input data prepared by the user however is modified to indicate data base inputs. The modified data input does not affect the applications program since the DIALOG executive program inspects the data input prior to execution of the applications program. DIALOG combines the required data base information with the basic program inputs, then prepares automatically a file containing the modified input format for the applications program and provides for the execution of that program in the nominal manner. This is illustrated schematically in Figure 3-11.

It should be noted that the applications program may still be executed in the normal manner as a stand-alone program independent of the DIALOG executive system.

3.2.3.2 Data Base Communication through Input

Data base information is entered into the applications program input by means of the special delimiters (‘). Any data base variable name may be entered between the delimiters. The DIALOG executive program will replace the variable name by its data base value and rewrite a

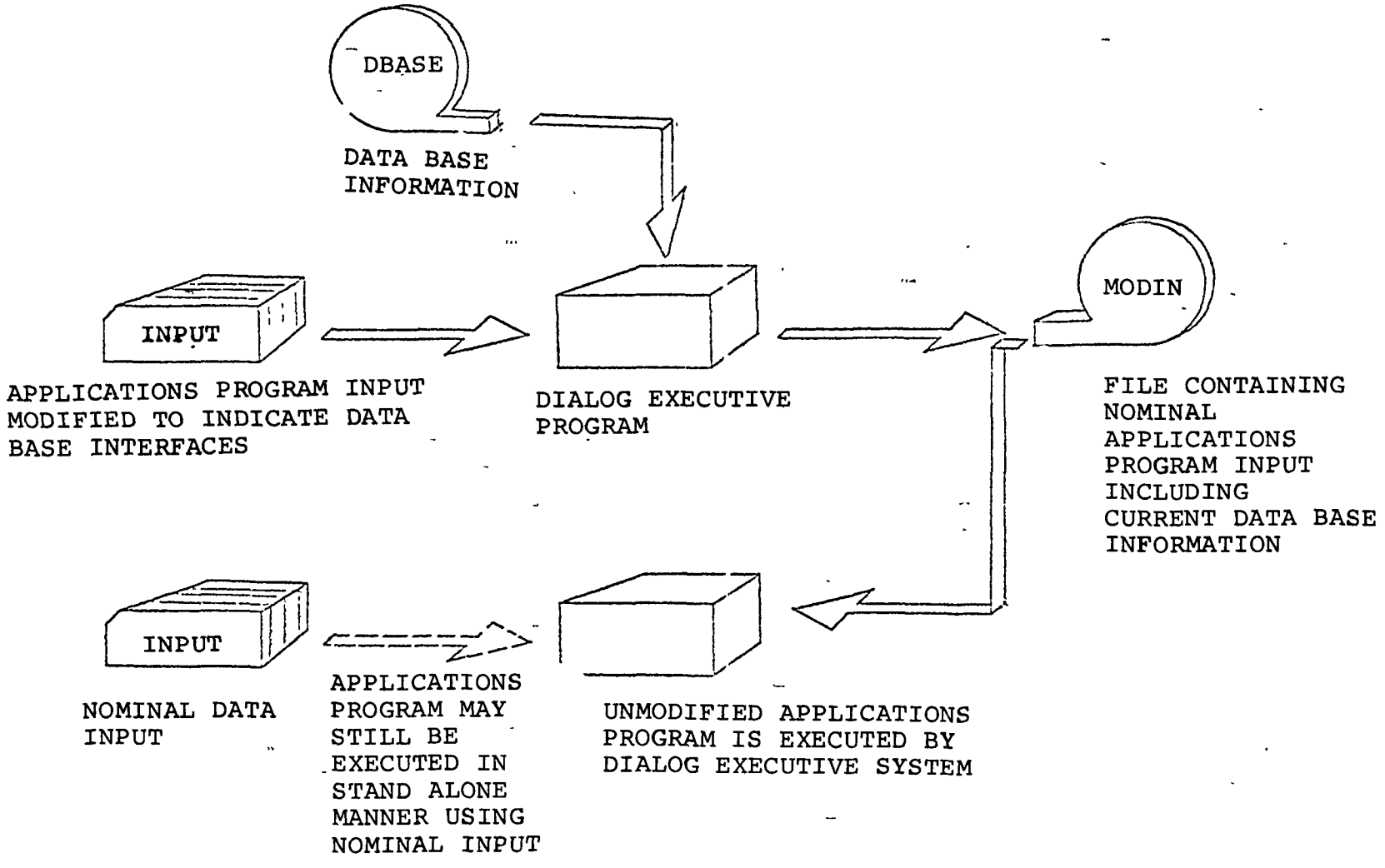


FIGURE 3-11 METHOD FOR MODIFYING INPUT DATA WITHOUT PROGRAM MODIFICATION

normal card image to replace the modified input cards. The value is placed within the closed region which includes the delimiters. Therefore, namelist-like inputs, rigid format input and special input procedures can be accommodated by the general input modification. For example, in a true namelist input, a data base variable would be entered as follows:

```
NAM1 = 'V1',
```

NAM1 is the name of the NAMELIST variable. V1 is the data base name. The delimiters specify the field width to be employed in replacing the data base name with the corresponding data base value. Similarly for a formatted input where all that normally appears on a card is a number the data base input procedure is simply:

```
'V1'
```

The delimiters are placed at the appropriate card columns defining the field for the data element.

Additional capability is available when namelist input is used by the applications program. Entire arrays may be transferred to the input stream.

```
NAME = 'VARRAY'
```

where VARRAY is a data base array. If the data in VARRAY is more than three elements, additional 'cards' are created to pass all the information in the VARRAY to the input stream of the applications program.

3.2.3.3 Combining Data Base Information in the Modified Input Stream

Data base variables and constants may be combined much like the capability described for the ADD command in Section 3.2.2.2.

For example, the operation may be performed:

```
'V1 * V2'
```

The above example illustrates how the multiplication of data base variable V1 by data base variable V2 can be performed prior to replacing the delimited set with the product of the multiplication. A new data base variable representing the combination is not created. The product never resides in the data base, only in the modified input stream. V1 must be a data base variable but V2 may be a data base variable or constant.

The operation illustrated above indicates a multiplication of the two numbers enclosed in delimiters prior to replacement of the delimited command with the product. Any algebraic operator may be employed.

```
 + addition
 - substraction
 * multiplication
 / division
** exponentiation
```

More than one of the above operations may be performed within the delimiters.

```
'V1 + V2 * V3'
```

Up to ten operations may be performed within a single command. The hierarchy of operations is the same as that described for the ADD command.

In general, array elements may be used in the replacement command:

```
'V1(5)' or
'V1(5) * V2(6)'
```

One exclusion from this capability is the first element of an array:

```
'V1(1)'
```

The above illustration is not an acceptable statement to the DIALOG executive for transferring the first element. Any other element of any array may be employed. A convenient means of avoiding the above limitation is the following two cards:

```
'ADD NEW = V1(1)'

'NEW'
```

The ADD command defines a new location which will contain V1(1) the replacement command will place the variable NEW (i.e. V1(1)) in the modified input stream.

A special feature of the replacement command is the element-by-element combining of entire arrays or arrays with constants. As an illustration, consider the example:

```
'V1 * V2'
```

where the data base variables V1 and V2 are arrays. The above command specifies the element-by-element multiplication of the arrays. If one array has fewer elements than the other, the combining of elements ceases after the shorter array is exhausted and the rest of the longer array remains unchanged. The variable V2 may be a constant or data base name.

3.2.4 Communicating Information from the Applications Programs to the Data Base

The communication of information from an applications program to the data base generally, but not always, requires modification of that

applications program. This modification is usually trivial and involves little programming knowledge to accomplish. Further, a mechanism is available within the FORTRAN language which further simplifies the task. This mechanism is the NAMELIST output feature. A more detailed description of the modification procedure is described in Section 4. The objective of the modification is to generate a special file of information available to the DIALOG executive system, which contains the desired information in the proper format.

The special file is interrogated by DIALOG for name oriented information in the following format:

```
$name name = value, value, $
```

Note the similarity between the above format and the FORTRAN NAMELIST feature.

In generating the file within the applications program, the analyst has the option of using NAMELIST (in FORTRAN programs only) or simply simulating NAMELIST by following four basic rules.

a. Separate name and values with equal (=) signs (N = V1).

b. Separate name/value sets with commas (,) (N1 = V1, N2 = V2).

c. Separate values (as in an array) with commas (,) (N3 = V1, V2).

d. Delimit multiple name value sets with a delimited ADD like "command" ($name.....$).

4.0 INSTALLATION OF THE DIALOG EXECUTIVE SYSTEM ON A TYPICAL 6000 SERIES COMPUTER AND THE LIBRARY OF INDEPENDENT ANALYSIS PROGRAMS

The DIALOG executive system can be installed on any CDC 6600 computer which has an operating system containing control card linking capability. The linking capability is provided by an operating system utility program which directs the system to read control cards from a user specified file other than INPUT.

For computer installations not having this capability, two alternate system utility programs are included in the basic library which provide control card linking capability. At Langley Research Center (LRC), the control card linking utility installed is called CCLINK. Similar utilities are available at other installations where the DIALOG executive system is in use.

The description of the installation of the DIALOG executive system is strongly dependent upon the computer operating system in use. Each computer complex has unique operating system features not generally available at other computer complexes. System differences are reflected in the control cards.

As a specific example, the discussion in this section is directed toward the installation of the DIALOG executive system at Langley Research Center. Therefore, the control cards described (including CCLINK) will be peculiar to the operating system employed at the LRC facility. Appendix A summarizes the control cards commonly used in the DIALOG executive system at the LRC computer complex. Complete detailed information for control card usage at LRC may be obtained from literature available upon request from the LRC computer complex.

Before installing the DIALOG executive system the storage devices for the independent programs must be chosen. This affects both the procedure in storing the program library as well as the entries into the control card data base. The program storage devices can be either data cell (online) or tape (offline) at LRC. The two device types may be mixed if so desired. However, the data cell is the most commonly used at LRC and considering the number of programs involved, the use of data cells assures a smoother operating system. Therefore, data cell storage will be discussed exclusively in this section.

In general terms the installation of the DIALOG executive system involves four basic tasks:

a. Compile and store the DIALOG executive programs.

b. Compile and store a library of independent programs.

c. Create a control card data base containing the control card sequences of each independent program.

d. Store the DIALOG executive system including the control card data base.

When all of the above tasks have been accomplished, the use of the DIALOG executive system may commence as described in Section 5. The above tasks will be described in detail in this section.

4.1 Compilation and Storage of the DIALOG Executive Program

The objective of this paragraph is to describe first the compilation and storage of DIALOG, a brief description of how DIALOG fits into the DIALOG executive system, a discussion of data base parameters and finally a deck set up for storing DIALOG.

The DIALOG executive has two main programs:

DBINIT - initialization

DIALOG - language processing

DBINIT is the DIALOG executive system initialization program. DBINIT is called only once when a new design data base (see Section 5) is being created, but performs no language processing function. DIALOG is the language processing program. DIALOG is executed initially to establish the execution sequence of the independent programs; then is called after the execution of each independent program to process the data for the independent programs.

4.1.1 Data Base Parameters

Before compiling the executive programs the design data base size must be established through the data base parameters. These parameters control the number of names and number of data elements which can be stored in the dynamic data base as described in Section 3. They also control the length of the description for each entry. All data base parameters directly affect the core storage requirements for the DIALOG executive system.

As an example, for a data base size of 3000 data elements, 200 names and a description length of three computer words, the core storage requirement for the system is 55000 (base 8). The above parameters are the standard values for the DIALOG executive system. Unless there are specific requirements for the data base parameters, they may be left as they are. However, they are usually set such that the core size matches the core size of the largest independent program anticipated for the system or, as an example, they might be adjusted for remote terminal operation (70000 - base 8).

The alteration of the data base parameters requires changes to the following programs or subroutines:

DBINIT - data base initialization program

DIALOG - language processing program

DBLOAD - data base loader subroutine

INITL - DIALOG initialization subroutine

The data base consists of two parts; a packed array for storage of data elements and a directory of names and pointers to the data element locations. Both the number of names and number of data base elements may be altered. The directory also includes provisions for a description array for each name. The data base parameters involved in changing the data base size are:

DBLEN - length of the data base (i.e. the packed array of data elements)

DIRLEN - directory length or number of name entries possible in the directory

DIRWID - directory width

The directory width consists of the following elements or words:

a one word for the name

b one word for the value (pointer)

c one word for the "hash" table

d one word for the "collision" table

e one word for the update command

f a variable number of elements for the description

The minimum directory width is five which allows no space for descriptive information in the directory. Therefore, the variable DIRWID may be computed as follows:

DIRWID = 5 + number of words of description

Once the three data base parameters are established, the parameters and dimensions may be set in the four affected routines:

DBINIT:

```
COMMON /DTBASE/ IDATA(2*DBLEN)

COMMON /DIRECT/ IOPR(50), INUM(60), ID(DIRLEN*DIRWID)
```

DIALOG:

```
COMMON /DTBASE/ IDATA(2*DBLEN)

COMMON /DIRECT/ IOPR(50), INUM(60), ID(DIRLEN*DIRWID)
```

```
DBLOAD:

   COMMON /DTBASE/ IDATA(2*DBLEN)

   COMMON /DIRECT/ IOPR(50), INUM(60), ID(DIRLEN*DIRWID)

INITL:

   DATA KEYLEN /1/, DBLEN /DBLEN/, DIRLEN /DIRLEN/, DIRWID /DIRWID/
```

The underlined parameters are the only ones requiring alteration by the user.

4.1.2 Deck Setup for DIALOG Storage

The DIALOG executive programs are stored in source form and relocatable binary elements (LGO file) on a data cell. The control cards, excluding the "accounting" cards, are discussed in Appendix A. It is assumed the reader is familiar with the above mentioned "accounting" cards. (See LRC computer usage manuals for complete details.)

The deck setup for compiling and storing DIALOG is shown in Figure 4-1. This deck setup assumes space on data cell has been pre-assigned.

The sequence of utility operations is:

- copy the source code to SCFILE
- compile the source code
- store the source code and binary elements

SCFILE and LGO are stored on a new data cell using the system utility STASH. All programs stored on data cell must have a "label" supplied by the LRC computer operations personnel. This label is read by the system utility STASH as input data. The label must be read by STASH or data cell storage cannot be accomplished.

4.2 Compilation and Storage of a Library of Programs

Since the DIALOG executive controls the sequence of execution of a library of independent programs, each program in the library must be stored in the computer system. The library includes not only user supplied programs but also system programs such as the compiler and the data cell storage and retrieval programs.

The manner in which the programs are stored is analogous to the storage of the DIALOG program depicted in Figure 4-1. However, there are two additional points to consider in preparing an analysis program for use with the DIALOG executive system:

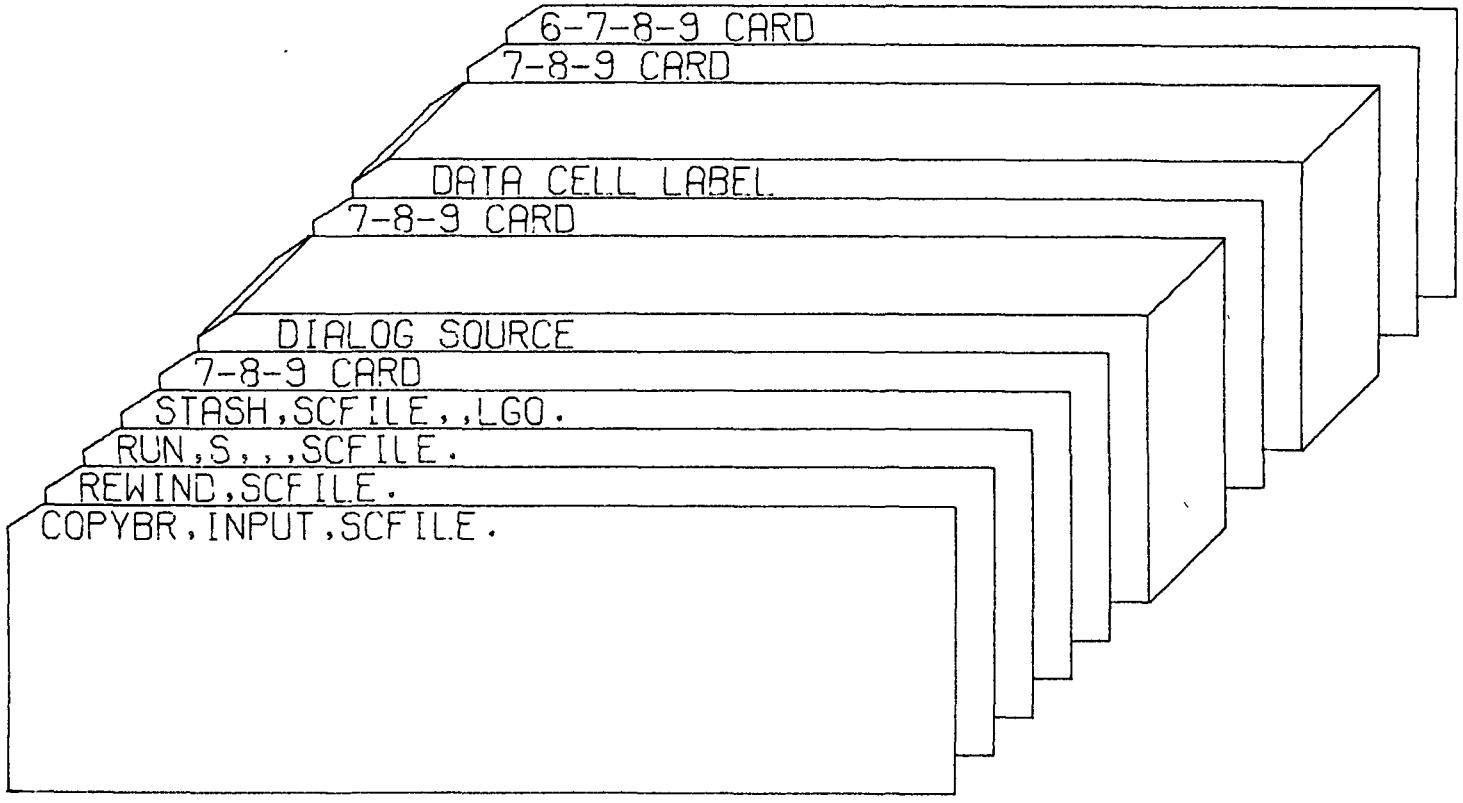


FIGURE 4-1 DECK SETUP FOR COMPILING AND STORING DIALOG

a. The program may require modification to provide a special output file of data base information as briefly mentioned in Section 3.

b. The program may be stored in absolute element form for high speed loading and reduced core requirements.

4.2.1 Program Modification to Provide Data Base Information

The communication of information from an applications program to the data base generally, but not always, requires modification of that applications program. There are a number of analysis programs available which generate files of information suitable for use with the DIALOG executive system, either directly or through an interface program. An interface program is often the most convenient means of extracting data from an analysis program.

The interface program obtains its input from a file generated by another program. The output will be precisely as specified in the following paragraphs. In this sense, the interface program is simply another analysis program in the program library.

Whether the original program is modified or an interface program is written, the generation of the data base output file is usually trivial and involves little programming knowledge to accomplish. If FORTRAN is the program language, a mechanism is available which further simplifies the task. This mechanism is the NAMELIST output feature of FORTRAN. The use of NAMELIST is briefly discussed below.

4.2.1.1 Creating a Special Output File

The objective of the modification is to generate a special file of information available to the DIALOG executive system, which contains the desired information. This is accomplished with an additional file parameter on the program card of the applications program.

```
PROGRAM PGMA (INPUT, OUTPUT, TAPE78)
```

In the above illustration, the TAPE78 file is the special file which is read by the DIALOG executive program. The file number is optional and may be any available unit. The information placed in this file is interpretted and then transferred to the data base by the DIALOG executive program. The mechanism by which the file is passed from the applications program to the DIALOG executive, is referred to as file substitution. The file substitution is accomplished with the control cards stored in the control card data base. As an illustration of file substitution technique, consider the execution of PGMA above. The execution control card for PGMA would be:

```
PGMA, MODIN, OUT, NMLIST.
```

The above card specifies the execution of PGMA. Additionally, it specifies that the INPUT file for PGMA will be MODIN, the OUTPUT file will be system file name OUT and the TAPE78 file will be the system file name NMLIST. These files

are addressed internal to PGMA by the names on the program card, but addressed externally by the system file name. They are sometimes referred to as internal and external names respectively. DIALOG recognizes NMLIST as a file containing potential data base information. In a sense, NMLIST is an input file to DIALOG.

4.2.1.2 Format of the Special Output File

NMLIST is interregated by DIALOG for name oriented information in the following format:

```
$name name = value, value, $
```

Note the similarity between the above format and the ADD command described in Section 3.2.1 Also it is identical with the FORTRAN NAMELIST feature.

In generating the NMLIST file within the applications program, the analyst has the option of using NAMELIST (in FORTRAN programs only) or simply simulating NAMELIST as discussed in Section 3.

Generally the name selected is one which is similar to the program name such as:

```
$PGMAOUT
```

for PGMA output. The advantage of using this naming convention is apparent in the actual data base construction. The name PGMAOUT, is stored in the data base identifying which program or "command" which last updated the value or values stored.

4.2.1.3 Use of the NAMELIST Feature in FORTRAN

If the applications program is written in FORTRAN, the analyst can use the NAMELIST feature to write the special data base output file. For example, to transfer the variables ANAME, BNAME, CNAME, I1, I2, JNAME and these variable values to the data base, the following modification is required at the exit point:

```
NAMELIST/PGMAOUT/ANAME, BNAME, CNAME, I1, I2, JNAME

WRITE (78, PGMAOUT)
```

The format of TAPE78 for the above illustration is shown in Figure 4-2. The DIALOG executive program interrogates unit 78 after the execution of the applications program to find variable names and values to be entered into the data base.

4.2.2 Storage of an Absolute Element Program

In performing a given analysis, the applications program user usually compiles and loads the relocatable (LGO) binary elements. These operations require the use of the operating system compiler, RUN and the operating system loader, LOAD in the following manner:

```
$PGMAOUT
 ANAME = VALUE,
 BNAME = VALUE,
 CNAME = VALUE,
 I1    = VALUE,
 I2    = VALUE,
 JNAME = VALUE,
$END
```

FIGURE 4-2 FORMAT OF THE NMLIST OUTPUT FILE

RUN, S.

LGO.

The standard loader must reside in core while it loads the LGO file. Further, the loader requires an additional space allocation for "loader tables." The additional space requirements depend upon the particular program involved, but the total space for loader and tables varies from 5000 (base 8) to 15000 (base 8). The space required by the loader must be added to the space required by the resulting applications programs. The total space for both is the central memory field length required for the job and must be specified on the job card. The field length is automatically reduced after the program is loaded (i.e. the loader space becomes available to the system for other uses).

The above method of analysis results in two disadvantages:

a. More central processing and peripheral processing time used to compile and/or load.

b. Job turn around suffers from the increased field length requirements.

The disadvantages illustrated above both result in reducations in efficiency and productivity of both computers and analysts. In the DIALOG executive system, most programs are compiled and stored as absolute elements. An absolute element program is one generated by the loader. It is a self contained program which includes all the system routines called for by that particular program. The advantages of storing the absolute element program are that it:

a. Reduces the field length requirement on the program card by the size of the loader plus loader tables. The program is loaded by the peripheral processor loader which takes no space in the central processor.

b. Considerably reduces the load time required for the program. All external references are resolved. This means all operating system routines called by the program are stored in the absolute element.

One disadvantage of storing an absolute element program is the data cell space requirements are increased by the space required for system routines. This is not considered to be a serious disadvantage. Another disadvantage is that updating of an absolute element program requires recompilation of the entire source code or the auxiliary storage of binary program elements. In the latter case only those elements requiring modification need be recompiled. Complete recompilation of modified programs is the usual procedure at LRC.

4.2.2.1 Absolute Element Files for Overlayed Programs

The generation of an absolute element requires a special control card procedure.

The procedure is discussed below. In overlayed programs, the absolute elements have the same file names as the overlay names. The absolute element program can be saved simply by copying the overlay files generated by the loader to a permanent storage device. The sequence of operations and the deck setup involved in compiling and storing the source code and an absolute element program on data cell are shown in Figure 4-3. In this illustration the program is not actually executed. It is located as a necessary operation in creating an absolute element program.

The OVL file name is the same name as the overlay name indicated. Duplicate names should not be used when naming overlay files. This can cause problems in creating and using the control card data base.

The deck setup shown in Figure 4-3 assumes the data cell has been assigned and never before used.

4.2.2.2 Absolute Elements Files for Unoverlayed Programs

The storage of an absolute element for an unoverlayed program can be accomplished by using the same procedure as described above simply by placing an overlay card as the first card in the deck. To the operating system loader, the simple addition of an overlay card makes an "overlay program" out of one which is not actually overlayed.

4.2.2.3 Creating Overlay Files Using AUTOLAY

AUTOLAY is a utility program provided as part of the DIALOG executive system which is used in the construction of overlay files from relocatable binary elements. It is not an operating system utility and therefore must be retrieved from data cell storage (FETCH) before it can be used.

The use of AUTOLAY permits the construction of a new program file from one to six library type files. The new program contains the selected main program or programs together with all the subroutines (external references) which the main program (s) call. The main program (s) are specified by the user in a text file read by AUTOLAY. The control card required to execute AUTOLAY is:

```
AUTOLAY (NEW, LIB1, LIB2, LIB3, LIB4, LIB5, LIB6)
```

After execution of AUTOLAY, the NEW file will contain the specified main programs and all the subroutines used by the main program. The NEW file is generated by searching the LIB files for the names programs and the external references. The LIB files are searched sequentially from left to right, resolving external references by selecting the first subroutine encountered which has the correct name. Subroutines which are not called are not placed on the NEW file. The LIB files may themselves be overlayed program files.

Figure 4-4 illustrates the construction of an overlay program from the LGO file. In the illustration, the program consists of a main program PGMA and two subprograms SUBPGMA and SUBPGMB. Any overlay structure may be used. The main program of each overlay is listed following the OVERLAY card. Block data

```
COPYBR, INPUT, SCFILE.
REWIND, SCFILE.
RUN, S,,,SCFILE.
LOAD, LGO, .
NOGO.
STASH, SCFILE,, OVL.          (OVERLAY FILE)
7-8-9
     OVERLAY (OVL, 0, 0)
                              (OVERLAY NAME)
     SOURCE CODE
7-8-9
     DATA CELL LABEL
6-7-8-9
```

FIGURE 4-3 COMPILATION AND STORAGE OF SOURCE AND AN ABSOLUTE ELEMENT PROGRAM

```
COPYBR, INPUT, SCFILE.
REWIND, SCFILE.
RUN, S,,,SCFILE.
FETCH, A___, SPR___ BINARY,, AUTOLAY.
AUTOLAY, NEW, LGO.
LOAD, NEW.
NOGO.
STASH, SCFILE,, OVL.
7-8-9

     SOURCE CODE

7-8-9
OVERLAY (OVL, 0, 0)
PGMA
BLKDATA
OVERLAY (OVL, 1, 0)
SUBPGMA
OVERLAY (OVL, 2, 0)
SUBPGMB
7-8-9

     DATA CELL LABEL

6-7-8-9
```

FIGURE 4-4 ILLUSTRATION OF THE USE OF AUTOLAY

programs must also be listed since they are not referenced by any program or subroutine. The user need not be concerned with the order the routines are placed in the source or LGO file since AUTOLAY will reorder them according to which overlay element they belong.

AUTOLAY is a particularly useful tool for structuring a new program or restructuring an old one. It completely eleminates the need to order source decks by overlay structure. More details on the use of AUOTLAY are given in Appendix A.

4.2.2.4 Updating Absolute Element Programs on Data Cell

The previous discussion assumed the analysis program was a new program being stored for the first time. The procedure for updating an existing program on data cell is illustrated in Figure 4-5. Here the FETCH utility is employed to retrieve the source code from data cell storage. Whether or not the absolute element program was previously stored, the illustrated procedure will store the absolute element and source program. The REPLACE utility is employed for this purpose.

4.3 Assembly of the Control Card Data Base

The control card data base, CCDATA, contains all the control card sequences required to retrieve and execute the library of independent programs. After the independent program is stored and prior to creation of the DIALOG executive system, the control card sequences must be assembled. It is also desirable to assemble a series of utility procedures for the purpose of data disposition. These procedures enhance the usability of the DIALOG executive system.

The creation of a control card data base is a function of the DIALOG executive. DIALOG reads the control card sequences from input cards and stores them in CCDATA. The delimited control card directive which creates the control card data base is:

```
'CREATE CCDATA'
```

One and only one space may appear between the words CREATE and CCDATA. This directive is followed by a file of information containing the control card sequences for the various applications programs and utility functions as illustrated in Figure 4-6. The first entry name must have an opening delimiter immediately before the name. A closing delimiter must appear after the last entry on a separate card as shown. The file must be terminated by a 7-8-9 card.

The control card sequences stored in the CCDATA data base generally include cards for both the retrieval and execution function. In queuing control card sequences, the above functions are generally split by the DIALOG executive. The retrieval function is performed at the beginning of the sequence and the execution function is performed when the EXECUTE directive is encountered.

```
FETCH, A0000, SPRA00, SOURCE.
RUN, S,,, SCFILE.
LOAD, LGO.
NOGO.
REPLACE, SCFILE, OVL.
7-8-9
     SOURCE UPDATES
7-8-9
     DATA CELL LABEL
6-7-8-9
```

FIGURE 4-5 UPDATE OF AN EXISTING ABSOLUTE ELEMENT PROGRAM ON DATA CELL

'CREATE CCDATA'

'PGMA = { CONTROL CARD SEQUENCE }

PGMB = { CONTROL CARD SEQUENCE }

PGMC = { CONTROL CARD SEQUENCE }

7-8-9

FIGURE 4-6 ILLUSTRATION OF A CONTROL CARD DATA BASE CREATION FILE

The reason for splitting the functions is to avoid multiple retrievals in the repetition of control card sequences. It avoids the repetitious retrieval of applications programs when control card sequences are repeated. Therefore the first card in a control card sequence is always the control card to retrieve the applications program. If no applications program is involved as in the execution of system programs only (i.e. the compiler) then the first card must be a system comment card.

The following paragraphs describe the formation of a single entry in the control card data base for a hypothetical applications program. Following this discussion, a set of standard utility procedures will be discussed.

4.3.1 Construction of a Control Card Sequence for a Data Base Entry

An illustration of a data base entry is shown in Figure 4-7. The essential features are:

a. A name for the control card sequence followed by an equal (=) sign.

b. A FETCH (retrieval card) is the first card in the sequence.

c. An execution card.

d. CCLINK, DIALOG.

The name is selected by the analyst at the time the control card entry is constructed. The name will be used to retrieve the control card sequence. The EXECUTE directive is used for this purpose.

'EXECUTE PGMA'

Common practice dictates the use of the acronym associated with an applications program such as AESOP for Automated Engineering and Analysis Program. Most applications programs have such an acronym associated with them.

Application program execution sequences require a FETCH card first. If no application program is being used in the execution sequence, the first card must be a comment card:

COMMENT.

Following the FETCH card any utility functions required by the applications program prior to execution may be specified. One example could be a REWIND for some file needed by the applications program.

The execution card begins with the overlay file name from the FETCH card. The remaining parameters are the file substitution parameters. In the illustration, the first three internal files are assumed to be INPUT, OUTPUT, and TAPE78 where TAPE78 is the special data base output file. These file names are (and must be) replaced on the execution card with MODIN, OUT,

```
PGMA =
FETCH, A0000, SPRA00, BINARY,, OVL.
- - - - -
          (OTHER PRE-EXECUTION CONTROL CARDS)
- - - - -
OVL, MODIN, OUT, NMLIST.
- - - - -
          (POST-EXECUTION POST CARDS)
- - - - -
CCLINK, DIALOG.
EXIT.
CCLINK, ABEND.
```

FIGURE 4-7 SAMPLE CCDATA ENTRY

NMLIST, as illustrated. MODIN is the modified input generated by DIALOG. OUT is the output storage file for the program. OUT is generally not printed except when the control directive:

'EXECUTE PRINTER'

is used. PRINTER is the name of a special control card procedure which prints the OUT file. Details of this and other procedures are discussed in Section 4.3.2.

NMLIST is the file containing data base output information.

Following the execution card any utility function required as a result of executing the applications program may be performed. An example could be the DROPFIL utility for releasing of scratch files generated by the applications program.

The last essential card is:

CCLINK, DIALOG.

This card causes the re-execution of the DIALOG executive programs at the termination of the control card sequence. Two additional cards which are not essential but usually desirable are:

EXIT.

CCLINK, ABEND.

These utilities are executed in case of a fatal error in the applications program.

Among other things, the ABEND procedure causes the OUT file to be printed. The ABEND procedure is discussed in detail in Section 4.4.

The construction of every control card procedure for each applications program follows the same pattern as described above. Variations can occur as a result of:

a. Different file parameter positions on the applications program PROGRAM card.

b. Possible use of utility control cards before and/or after the execution card.

There is no effectove limit to the number of control cards which can be stored by a given name. Once stored, however, the number of control cards cannot be changed without recreating the control card data base.

Appendix B is a table of control card sequences for a group of commonly used applications programs and utility procedures at LRC. This appendix

is an actual status report on the control card data base from the DIALOG executive system.

Any control card in any sequence may be altered temporarily or permanently during the execution sequence. The method by which this is accomplished is discussed in Section 5.

4.3.2 Standard Utility Sequences

During the development phase of the DIALOG executive system, a set of utility procedures evolved for performing such functions as:

a. Disposition of data files.

b. Compilation and execution of interface programs.

c. Executing arbitrary control card procedures.

These procedures are stored in the control card data base and are called into execution with the EXECUTE directive. The following paragraphs provide a brief discription of each standard utility procedure and its use. Appendix B gives a detailed list of the actual control card sequences for the Langley system.

4.4 Storage of the DIALOG Executive System

The final task in the installation of the DIALOG executive system is the storage of the executive programs and special procedures. When this task is accomplished, simulations can begin.

4.4.1 Elements of the DIALOG Executive System

The DIALOG executive system consists of two FORTRAN programs discussed in Section 4.1, the control card data base discussed in Section 4.2 and 4.3 and the following special procedures:

ODIN	DIALOG executive system initialization procedure
DIALOG	DIALOG executive program execution procedure
ABEND	Abnormal end procedure
PINIT	Post initialization procedure

Details of these procedures are described in Appendix C.

The entire system is stored on a data cell as source and binary records as illustrated in Figure 4-8.

The ODIN procedure is stored as source. The purpose is two fold.

a. Minimize the number of operating system control cards required to start a simulation.

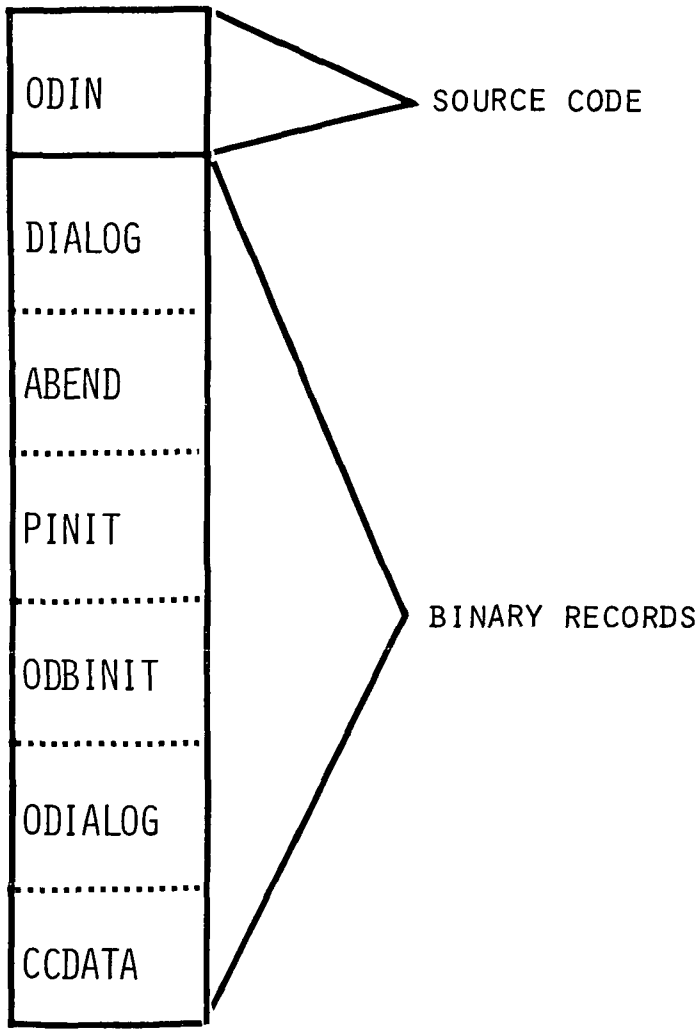


FIGURE 4-8 ILLUSTRATION OF THE DIALOG EXECUTIVE SYSTEM STORAGE FILE.

b. Provide a convenient means for modification of the initialization procedure (as in the use of a restart capability).

Two control cards (other than JOB and USER cards) are required to start a simulation.

```
FETCH, A----, SPR---.  BOTH, ODIN.

CCLINK, ODIN.
```

The ODIN procedure then 'boot straps' the rest of the DIALOG executive system in from the binary file associated with the data cell. Modification of the ODIN procedure may be accomplished by the usual methods at the LRC computer complex.

4.4.2 Deck Setup for Storing the DIALOG Executive System

Figures 4-9 and 4-10 are illustrations of the deck setup for storing the DIALOG executive system. Figure 4-9 presents the control card sequences. Figure 4-10 gives the data files. In actuality the deck setup presented creates several necessary files discussed in Section 4.4.1 and links to the DIALOG executive system. The steps in creating the system are discussed below.

4.4.2.1 Modification of the DIALOG Program

This step allows the analyst to make modifications to the DIALOG program by the FETCH procedure presented in Reference 14. One example of a DIALOG modification would be the alteration of the data base parameters as described in Section 4.1.1. If no modifications are desired, the DIALOG modification file in Figure 4-10 is left null (7-8-9 card only). Following modification of the DIALOG source code, the DIALOG program is compiled. A relocatable binary file (LGO) is created. This file will be used for generating the absolute element programs ODIALOG and ODBINIT.

4.4.2.2 Creation of a Data Cell Location for the DIALOG Executive

A data cell storage location is created by the FETCH card shown in Figure 4-9. The actual storage (REPLACE) is accomplished by the execution of CCSAVE described below. However, two files must be saved for use by the REPLACE utility. The files saved are DCNS and ZOUNDS.

4.4.2.3 Creation of the Special Procedures for the DIALOG Executive System

The special procedures for the DIALOG executive system are shown in Appendix C. These include the following procedures:

ODIN — Initialization procedure - to retrieve the DIALOG executive system programs and procedures and to perform the initial execution of the DIALOG executive programs.

DIALOG — DIALOG Execution Procedure - for the DIALOG executive program used after each applications program execution sequence.

```
JOB, 1, 70, 7000, 4000. - - - - - -
USER, - - - - - - - - -
FETCH, A3974, SPRA--, SOURCE.        (DIALOG Program)
RUN, S,,, SCFILE.                    (Compile DIALOG)
FETCH, A____, SPR___, BOTH,,,,X.     (DIALOG Executive System)
DROPFIL, SCFILE, BNFILE, X.
REWIND, DCNS, ZOUNDS.
COPYBF, DCNS, ODINSA1.
COPYBF, ZOUNDS, ODINSA2.
COPYBR, INPUT, ODIN.                 (Initialization Procedure)
COPYBR, INPUT, DIALOG.               (DIALOG Execution Procedure)
COPYBR, INPUT, ABEND.                (Abnormal End Procedure)
COPYBR, INPUT, PINIT.                (Post Initialization Procedure)
REWIND, ODIN, DIALOG, ABEND, PINIT.
FETCH, A3596, SPRZ04, BINARY,, AUTOLAY.  (Retrieve AUTOLAY)
AUTOLAY, DBIN, LGO.                  (Construct OBINIT)
DBIN.                                (Execute DBINIT)
DROPFIL, AUTOLAY, DBIN.
FETCH, A3596, SPRZ04, BINARY,, AUTOLAY.  (Retrieve AUTOLAY)
AUTOLAY, DLOG, LGO.
DLOG.                                (Execute DIALOG)
DROPFIL, DLOG. AUTOLAY, LGO.
CCLINK, NMLIST.
7-8-9
```

FIGURE 4-9 CONTROL CARDS FOR INSTALLATION OF DIALOG EXECUTIVE SYSTEM.

```
                DIALOG MODIFICATIONS
7-8-9
                (ODIN Initialization Procedure)
7-8-9
                (DIALOG Execution Procedure)
7-8-9
                (ABEND Procedure)
7-8-9
                (PINIT Procedure)
7-8-9
                (AUTOLAY Text Cards for DBINIT)
7-8-9
                (AUTOLAY Text Cards for DIALOG)
7-8-9
                'CREATE DBASE'
7-8-9
                'CREATE CCDATA'
                (CCDATA entries)
7-8-9
                'EXECUTE CCSAVE'
                CHOOSE 82100000     83800000
7-8-9
                'END ODIN'
6-7-8-9
```

FIGURE 4-10 DATA FILES FOR THE INSTALLATION OF THE DIALOG EXECUTIVE SYSTEM

ABEND Abnormal End Procedure - for printing the output file from the previous program and for saving CALCOMP plot files.

PINIT Post Initialization Procedure for eliminating unnecessary scratch files used in the initialization of the DIALOG executive system.

Each procedure is created on a separate file as illustrated in Figure 4-10 and retained in the system until the end of the job. Upon execution of the CCSAVE procedure, these files are saved together with the DIALOG executive programs on the data cell described in Section 4.4.2.2.

4.4.2.4 Creation of Absolute Element Programs and Initializating the DIALOG Executive System

Overlay elements for DIALOG and DBINIT are created with the special utility AUTOLAY described in Section 4.2.2.3. All the essential subroutines for both programs are stored in the LGO file described in Section 4.4.2.1.

The text cards indicated in Figure 4-10 for creating DBINIT are:

```
OVERLAY (ODBINIT, 0, 0)

DBINIT
```

Upon execution of the resulting program, an absolute element called ODBINIT is created (by the loader). This absolute element program is available to be saved on data cell when the CCSAVE procedure is executed. The execution of DBINIT (DBIN. in Figure 4-9) also initializes the DIALOG executive system. The text cards indicated in Figure 4-10 for creating DIALOG are:

```
OVERLAY (ODIALOG, 0, 0)

DIALOG
```

Upon execution of the resulting program, an absolute element called ODIALOG is created (by the loader). This absolute element program is available to be saved in data cell when the CCSAVE procedure is executed.

The execution of DIALOG (DLOG is Figure 4-9) also preprocesses the remaining data files in Figure 4-10. In effect, the DIALOG executive system has control and the following DIALOG functions are performed:

```
CREATE DBASE

CREATE CCDATA

EXECUTE CCSAVE

END ODIN
```

At the completion of the above sequence of control directives, the DIALOG executive system is stored on the data cell described in Section 4.4.2.2. Simulations using the DIALOG executive system may begin.

5.0 USE OF THE DIALOG EXECUTIVE SYSTEM

The discussion in Section 4 centered around the installation of the DIALOG executive system including a control card data base. The present discussion deals with the use of the DIALOG executive system with a library of independent applications programs. Figure 5-1 illustrates how the use of the DIALOG executive system is used. There are four basic steps in using the system. First the task must be defined, which includes a consideration of the technology areas to be included in the analysis. Further the depth of analysis in each technology area must be selected. The depth of analysis and subsequent program selection will have a direct bearing on the computer resources which will be required. The second task is the selection of the applications programs and the sequence which will be executed. This requires some understanding of the basic data required by each program and the information each program generates. The above task may be a team effort if the simulation is large and/or requires many programs.

A survey of the existing ODIN library programs will aid in the selection process. If a suitable set of programs is not available, one of the following procedures may be employed:

a. Locate the necessary programs from an outside source.

b. Develop the necessary programs to serve the purpose.

In either of the two above cases, the program modification for use with the DIALOG executive system as discussed in Section 4 may be required.

The third task is the definition of the interprogram data which will be stored in the data base. Included in this definition are the study parameters, those elements of data which drive the study either through a parametric variation or an optimization process. Additionally, the performance criteria must be defined. The performance criteria is the desired study output information such as the weight or cost of the system under study. Often there are study constraints, those conditions which must not be violated in an acceptable solution. The study constraints and performance criteria are the basic information used in guiding the selection of values for the study parameters.

Interprogram data must also be defined in the data base. These data include intermediate results produced by one applications program which are used by other applications programs. Finally the data base may include information which may be used for monitoring the study.

The fourth and final task is the actual deck setup. Deck setup is the task which is addressed in this section. The implication is that the other three tasks have been performed; the remaining objectives are to set up the normal data for the selected applications programs then insert the proper control directives and communication commands to perform

1. DEFINE THE SIMULATION TASK
   - TECHNOLOGY AREAS TO BE CONSIDERED
   - DEPTH OF ANALYSIS
2. SELECT THE APPLICATIONS PROGRAM SEQUENCE
   - SURVEY THE AVAILABLE PROGRAMS
   - UPDATE THE PROGRAM LIBRARY
   - DEFINE THE EXECUTION SEQUENCE
3. DEFINE THE DATA BASE INFORMATION
   - STUDY PARAMETERS
   - PERFORMANCE CRITERIA
   - STUDY CONSTRAINTS
   - LIBRARY INTERPROGRAM DATA
   - MONITORING INFORMATION
4. DECK SETUP
   - SETUP NORMAL DATA FOR ALL PROGRAMS
   - INSERT THE CONTROL DIRECTIVES
   - INSERT THE COMMUNICATION COMMANDS

FIGURE 5-1 PROCEDURE FOR THE USE OF THE DIALOG EXECUTIVE SYSTEM

the desired simulation. This section is divided into discussions of control directives and communication commands. Control directives are instructions to DIALOG for the control of the sequence of execution of the independent applications programs. Communication commands are instructions to DIALOG for the merging of data base information with applications program data. Finally, the use of standard utility procedures will be discussed.

5.1 Control Directives

The use of the DIALOG executive system requires that the DIALOG executive programs and procedures be available on disk before the DIALOG control directives can be employed. Figure 5-2 is an illustration of the operating system control cards at LRC required to access the DIALOG executive system. Following the four control cards shown, all remaining information is processed by the DIALOG executive. As such the rules governing the use of the DIALOG executive system apply.

The general arrangement of the input data to the DIALOG executive system is shown in Figure 5-3. Here a hypothetical simulation is setup which illustrates the use of all control directives and alternates. Each control directive in this illustration will be described in detail. Generally speaking, the control directives have the following format:

```
'directive name'
```

The directive is enclosed by DIALOG delimiters ('). No space is permitted between the first delimiter and the directive. One and only one space is permitted between the directive and the name.

5.1.1 CREATE Directive

Initially the CREATE directive is used to establish a design data base.

```
'CREATE DBASE'

    file of data
    to initially
    establish DBASE

7-8-9
```

The CREATE directive is followed by a file of data to initially establish the design data base. Initial data is not essential to the operation of the system as data may be added at any point in the execution sequence. However, the 7-8-9 end-of-file mark is required regardless of whether data is entered. Data is always entered via the communication commands described in Section 5.2.

The CREATE directive is also used to create a control card data base.

JOB, 1, 70, 56000, 2000, _____

USER, _____

FETCH, A _ _ _, SPR _ _ _, BOTH, ODIN,, X.

CCLINK, ODIN.

7-8-9

{ DIALOG CONTROL DIRECTIVES } DIALOG INPUT

6-7-8-9

NOTE: THE CONTROL CARDS ILLUSTRATED ARE THOSE USED AT LANGLEY RESEARCH CENTER. OTHER INSTALLATIONS USE A DIFFERENT SEQUENCE BUT PERFORM THE SAME FUNCTION.

FIGURE 5-2 SYSTEM CONTROL CARDS REQUIRED TO ACCESS THE DIALOG EXECUTIVE SYSTEM.

```
                                        'RESTART'
'CREATE DBASE'          OR              'UPDATE DBASE'
        [FILE OF DATA INPUT TO THE DESIGN DATA BASE]
7-8-9

'UPDATE CCDATA'         OR              'CREATE CCDATA'
        [FILE OF DATA INPUT TO THE CENTRAL CARD DATA BASE]
7-8-9

'DESIGN START'          (OPTIONAL DESIGN IDENTIFIER)

'EXECUTE PGMA'
        [FILE OF INPUT DATA FOR PGMA]
7-8-9

'LOOP TO START'         (CONDITIONAL BRANCHING LOGIC)
'IF V1. LT. V2'
7-8-9

'EXECUTE PGMB'
        [FILE OF INPUT DATA FOR PGMB]
7-8-9

'PRINT DBASE'           (OPTIONAL PRINT COMMAND)

'END'

6-7-8-9
```

FIGURE 5-3 EXAMPLE OF A HYPOTHETICAL EXECUTION SEQUENCE ILLUSTRATING THE USE OF ALL CONTROL DIRECTIVES.

```
'CREATE CCDATA'

    file of control
    card data base
    entries

7-8-9
```

The control directive illustrated above creates a new control card data base, overwriting any existing one. The creation of a control card data base is described in detail in Section 4. Usually this directive is not used in a routine simulation since a control card data base already exists.

5.1.2 RESTART Directive

The CREATE DBASE directive must be the first directive in the sequence unless the simulation is the continuation of an earlier run. In the latter case, the following directive is used in place of CREATE DBASE:

```
'RESTART'
```

For the illustrated command to be effective, the data base created in the earlier run must have been saved by use by the ENDODN procedure described below. Further, the DIALOG executive system initialization procedure differs in case of a restart. Figure 5-4 illustrates the restart procedure. Here the execution of DBINIT (See Appendix C) is replaced by a FETCH control card for the retrieval of the previous design data base.

5.1.3 UPDATE Directive

The UPDATE directive is used for updating existing information in the data base and has the following format:

```
'UPDATE name'

    file of data
    to update existing
    data base

7-8-9
```

The name associated with the illustrated directive can be DBASE or CCDATA. The directive may be used for DBASE after a RESTART directive for updating information in an existing design data base. Often the UPDATE directive is used with CCDATA to alter an applications program procedure. A card-by-card update may be performed in the following manner:

```
'UPDATE CCDATA'
```

Following this control directive, the analyst lists the desired modifications.

```
JOB, 1, 70, 56000, 2000. ---
USER, ---
FETCH, A----, SPR---, BOTH, ODIN.
CCLINK, ODIN.
7-8-9
CUTOUT          1400000   (COLUMNS 14-20)
FETCH, A----, SPR---, BINARY,, DBASE
7-8-9
        { DIALOG    }
        { CONTROL   }
        { DIRECTIVES}
6-7-8-9
```

FIGURE 5-4 ILLUSTRATION OF THE RESTART PROCEDURE

```
PGMA =

new control
card
sequence
```

The above example illustrates the complete replacement of a control card sequence. If however, the analyst desired to change only one card in the sequence, the following cards might be employed:

```
PGMA (3) =

replacement control card
```

The above example illustrates the replacement of the third card in the sequence. The third and fourth card could be replaced in the following manner:

```
PGMA (3) =

replacement for third card
replacement for fourth card
```

The requirement for updating the control card data base stems from the fact the library programs are often modified or replaced as a result of revisions or new versions. Often more than one version of the same program exists. This is particularly true during periods of program development. During these periods of development, the analyst may wish to evaluate a test version of a program in the DIALOG executive system. The UPDATE directive affords the individual analyst the opportunity to make temporary control card modifications to the DIALOG executive system without affecting the other users of the system.

5.1.4 DESIGN Directive

The DESIGN directive is used to establish a point in the execution sequence to which control may be returned (or skipped to) via a LOOP TO directive described below. The DESIGN directive is analogous to a statement label in a FORTRAN program. The format of the DESIGN directive is:

```
'DESIGN name'
```

The name may be any name up to 10 characters but must not duplicate an existing design data base name.

5.1.5 EXECUTE Directive

The EXECUTE directive is used to execute a sequence of control cards from the control card data base. The format is:

```
'EXECUTE name'

     data file for the
     applications program

7-8-9
```

The name may be any name assigned to a control card sequence in the control card data base. This is the basic command for the execution of applications programs. The data file is the data for the applications program involved. The 7-8-9 card is required even if no data is involved with the execution.

5.1.6 LOOP TO Directive

The LOOP TO directive is used to transfer control to another point in the execution sequence.

```
'LOOP TO name'
```

In the above illustration, control is transferred to name. Name must be the name associated with a DESIGN directive described in Section 5.1.4.

5.1.7 IF Directive

The IF directive is a conditional branching directive used in conjunction with the LOOP TO directive in the following manner:

```
'DESIGN START'

 - - - - -

 - - - - -

'LOOP TO START'

'IF V1 . LT . V2'

7-8-9

 - - - - -

 - - - - -
```

In the above illustration, control is transferred to START if V1 is less than V2. V1 and V2 are design data base variables or constants. Multiple IF directives may be associated with any LOOP TO directive. When multiple IF directives are used, any condition specified will trigger the LOOP directive. If no conditional IF directives are used, the LOOP TO directive is mandatory. A 7-8-9 card is recommended after LOOP TO/IF directive sets.

5.1.8 PRINT Directive

The PRINT directive is used to print the design and control card data bases DBASE and CCDATA. The format is:

```
'PRINT DBASE'
```

or

```
'PRINT CCDATA'
```

This command may be placed at any point in the execution sequence but cannot be intermixed with a file of applications program data.

5.1.9 END Directive

The END directive is used to signify the end of a simulation. It has the following format:

```
'END   '
```

All simulations must be terminated with the END directive. A summary of the control directives is given in Appendix D.

5.2 Communication Commands

The communication commands provide a means of transferring information to and from the data base. The following types of transfer are included:

a. Transfer of information from the analyst to the data base.

b. Transfer of information from the data base to the applications program.

c. Transfer of information from the applications programs to the data base.

In the transfer of information from the analyst to the data base, three commands are used:

a. ADD command for adding or updating information in the data base.

b. DEFINE command for defining data base variables and reserving space in the data base.

c. . (comment) command for identifying applications program data.

In the transfer of information from the data base to the applications program, the basic command is the replacement command for replacing data base names and data base name/value combinations with values from the data base.

In the transfer of information from the applications program to the data base, a special extension of the ADD command is employed. The "ADD-like" command is used in generating the special output file described in Section 4.

Each of the above commands will be described in detail in this section. Examples covering most objectives will be presented.

5.2.1 The ADD Command

The ADD command is the basic communication command available to the analyst for adding or updating information in the data base. The general format of the command is:

```
'ADD name 1 = value 1, name 2 = value 2,

     name 3 = value 3, value 5, value 5,

              value 6, name 4 = value 7,'
```

It may be used in creating data in the data base as well as for modifying the data base at any point in the execution sequence. The ADD command must appear within the file of information following one of the control directives:

```
'CREATE DBASE'

'UPDATE DBASE'

'EXECUTE name'
```

The ADD command may be used for adding any type of information to the data base (i.e. real, integer, hollerith or logical) the data type being determined by the actual data entry. The information may be a single element or an entire array of information. Arrays may be of mixed types under certain circumstances. Combinations of data base variables and constants using the five common operators (+, -, *, /, **) may also be employed within the ADD command. However, mixed mode arithmetic is not permitted in combining data base variables. Specific rules and exceptions will be presented in the examples below.

The following list of rules of syntax or pattern of construction apply to the ADD Command:

1. The opening delimiter (') may be in any column.
2. No spaces may appear in the character string 'ADD (including the delimiter).
3. One or more spaces must appear between the ADD and the first name.
4. One or more name = value combinations may appear on each card.
5. The 'ADD command may be continued from card to card.
6. A card must be terminated with a comma (,) or a DIALOG delimiter (').
7. A continuation card may start with a name or value.

8. The 'ADD command must be terminated with a DIALOG delimiter (').

9. The closing delimiter may be on a separate card.

10. The last comma (,) before the closing delimiter is optional.

11. The maximum number of names and values (including the command) is 20 per ADD command.

The most common SYNTAX errors result from failure to comply with rules 2, 6 and 8. Attention is specifically drawn to these rules.

The following pages present some examples. Each example is discussed. In addition, a sample data base printout for all examples is given in Figure 5-6. Discussion of the examples is presented in the following format:

OBJECTIVE:	A concise statement of the analysis goal to be achieved by the command.
SYNTAX:	Pattern of formation of the command.
EXAMPLES:	A list of one or more examples. Sometimes incorrect examples are given and are noted as ILLEGAL. In these cases the results are either incorrect or unpredictable for the objective stated.
RESULT:	A brief description of the data base entry which will result from the use of the exemplified command.
RESTRICTIONS:	In some cases restrictions in addition to those described above will be given.

All of the ADD command examples presented in this section are illustrated in Figure 5-5. These examples are listed from cards which were actually processed by the DIALOG executive program. The delimiter (≠) shown in the illustration results from the particular character set employed by the printing device. The data base entries which resulted from the examples in Figure 5-5 are illustrated in Figure 5-6.

5.2.1.1 Adding Fixed Element Information

OBJECTIVE:	To add or update elements of information to the data base.
SYNTAX:	'ADD name = value'
EXAMPLES:	'ADD A = 10.'
	'ADD B = 2'
	'ADD C = ALPHA'

```
≠ ADD STATEMENT EXAMPLES ≠

≠ADD A = 10. ≠
≠ADD B = 2 ≠
≠ADD
      C =ALPHA ≠
≠ADD D = .TRUE. ≠
≠ADD E = A ≠
≠ADD F = B ≠
≠ADD G = A * 2. ≠
≠ADD H = B + 3 ≠
≠ADD I = A + G * 2. ≠
≠ADD J = 10., 20., 30. ≠
≠ADD K = 2, 4, 6 ,
      L = ALPHA, BETA, GAMMA ≠
≠ADD
      M = .TRUE., 3 , 9. ,
≠
≠ADD N = 2*8., 9., 9. ≠
≠ADD O = 2, 5., 6. ≠
≠ADD P = J(1) ≠
≠ADD Q = K(2) ≠
≠ADD R = A*O(3) ≠
≠ADD S = J(3) + O(2) ≠
```

NOTE: THE DELIMITER (≠) RESULTS FROM THE LISTING EQUIPMENT USED.

FIGURE 5-5 ADD COMMAND SUMMARY

NAME	LOCATION	DIMENSION	CURRENT VALUE(S)	ORIGIN	DESCRIPTION
A	1	1	10.	ADD	REAL VALUE
B	3	1	2	ADD	INTEGER VALUE
C	5	1	ALPHA	ADD	HOLLERITH VALUE
D	7	1	.TRUE.	ADD	LOGICAL VALUE
E	9	1	10.00000000000000000	ADD	E=A
F	11	1	2	ADD	F=B
G	13	1	20.00000000000000000	ADD	G=A*2.
H	15	1	5	ADD	H=B+3
I	17	1	60.00000000000000000	ADD	I=A+G*2.
J	19	3	10. 20. 30.	ADD	REAL ARRAY
K	25	3	2 4 6	ADD	INTEGER ARRAY
L	31	3	ALPHA BETA GAMMA	ADD	HOLLERITH ARRAY
M	37	3	.TRUE. 3 9.	ADD	MIXED LOG-INT-REAL ARRAY
N	43	4	8.000000000000000000 8.000000000000000000 9. 9.	ADD	SHORTHAND LOAD
O	51	3	2 5. 6.	ADD	MIXED TYPE ARRAY
P	57	1	10.00000000000000000	ADD	P=J(1)
Q	59	1	4	ADD	Q=K(2)
R	61	1	60.00000000000000000	ADD	R=A*O(3)
S	63	1	35.00000000000000000	ADD	S=J(3)+O(2)

FIGURE 5-6 DATA BASE ENTRIES RESULTING FROM THE EXAMPLE ADD COMMAND.

'ADD D = .TRUE.'

RESULT: Four data base locations will be created or updated in the data base. Each of the four basic data types are represented. The data type is determined from the actual number. There is no implicit type assumed from the data base name (as in FORTRAN).

ADDITIONAL RESTRICTIONS:

1. Hollerith information and logical variables which are to be stored in the data base cannot be data base names.

5.2.1.2 Adding Multiple Data Elements

OBJECTIVE: To add or update a series of elements to the data base by a single ADD command.

SYNTAX: 'ADD name 1 = value 1, name 2 - value 2'

EXAMPLES:

```
'ADD A = 10., B = 2,
     C = ALPHA,        See Restrictions
     D = .TRUE.'
```

RESULT: Four data base locations will be created or updated in the data base. The value stored will be those presented by the information to the right of each equal (=) sign.

ADDITIONAL RESTRICTIONS:

1. Hollerith information and logical variables which are to be stored in the data base cannot be a data base name.

5.2.1.3 Transferring Data Elements

OBJECTIVE: To create or update a data base variable by equating (transferring) one variable name to another.

SYNTAX: 'ADD name = name'

EXAMPLES:

```
'ADD E = A'
'ADD F = B'
```

RESULT: The variables E and F are created or updated. The information (real or integer) is transferred from A and B to the new locations for E and F.

ADDITIONAL RESTRICTIONS:

1. A and B cannot be hollerith or logical variables.
2. Neither E nor F can be an array name.

5.2.1.4 Combining Data Elements with Constants

OBJECTIVE: To create or update a data base variable by combining a data base variable with a constant using an arithmetic operation.

SYNTAX: 'ADD name = name * value'

EXAMPLES: 'ADD G = A * 2.' (real)

'ADD H = B + 3' (integer)

'ADD G = 2. * A' ILLEGAL - See Restrictions

RESULT: Two variables, G and H, will be created or updated in the data base whose values will be the combinations indicated. Any of the common arithmetic (+, -, *, /, **) operations may be performed. Up to ten operations may be performed.

ADDITIONAL RESTRICTIONS:

1. The combination of a variable with a constant cannot begin with a constant.

2. Neither G nor H can be an array name.

5.2.1.5 Combining Data Elements with other Data Elements and Constants

OBJECTIVE: To create or update a data base variable by combining other data base variables and constants.

SYNTAX: 'ADD name = name + name * value'

EXAMPLE: 'ADD I = A + G * 2'

RESULT: The contents of A will be added to the contents of G. The results of that combination will be multiplied by 2. The result of that combination will be transferred to I. Up to ten operations may be performed.

ADDITIONAL RESTRICTIONS:

1. The operations are serial in nature (like a hand calculator). Each operation is performed on the result of the previous operation (s).

2. The combination of variables must begin with a name.

3. The name, I cannot be an array name.

5.2.1.6 Adding Arrays

OBJECTIVE: To create or update arrays of information in the data base.

SYNTAX: 'ADD name = value, value, value'

EXAMPLES: 'ADD J = 10., 20., 30.,

K = 2, 4, 6,

L = ALPHA, BETA, GAMMA, (hollerith constants)

M = .TRUE., 3, 5.'

RESULT: Four new or existing arrays will contain the information indicated. Mixed arrays of real and integer values may also be used. Logical arrays (more than one element) cannot be entered.

ADDITIONAL RESTRICTIONS:

1. No data base name may appear on the right side of the equation (= sign) as array elements.
2. Logical and hollerith variables may not be mixed with any other type.
3. Hollerith may not be the first entry of an ADD card. (i.e., 'ADD L = ALPHA, BETA, GAMMA,)
4. No more than one logical variable per array may be entered.

5.2.1.7 Adding Constant Arrays

OBJECTIVE: To add or update an array of constant information to the data base.

SYNTAX: 'ADD name = n * value, value'

EXAMPLES: 'ADD N = 2 * 8., 9., 9.'

'ADD N = 8., 8., 2 * 9.' (ILLEGAL)

RESULT: N will be a new or updated array of four elements, each containing values of 8., 8., and 9., 9.

ADDITIONAL RESTRICTIONS:

1. The integer n must be next to the equal (=) sign. Multiple entries can only be performed on the first n elements.

5.2.1.8 Adding Mixed Arrays

OBJECTIVE: To add or update a mixed integer/real array of information.

SYNTAX: 'ADD name = value, value, ...'

EXAMPLE: 'ADD Ø = 2, 5., 6.

RESULT: The Ø array will be added or updated. It will contain the integer 2 followed by the real values 5., and 6. This is useful in some applications programs for defining table sizes in the same array as the tabular data.

ADDITIONAL RESTRICTIONS:

1. The array elements cannot contain hollerith or logical information with the integer and real information.

5.2.1.9 Transferring Array Elements

OBJECTIVE: To update or create a data base variable from an element of a data base array.

SYNTAX: 'ADD name = name (n)'

EXAMPLE:

```
'ADD P = J(1),
     Q = K(2), '

'ADD Q = K(B),'      (ILLEGAL)
```

RESULT: The contents of J(1) and K(2) are transferred to P and Q respectively. The element number being transferred must be a constant.

ADDITIONAL RESTRICTIONS:

1. The element number cannot be a data base name.
2. The element cannot be hollerith or logical in type.

5.2.1.10 Combining Array Elements

OBJECTIVE: To create or update a data base variable combining data base array elements.

SYNTAX: 'ADD name = name (n) * name (n)

EXAMPLES:

```
'ADD R = A * Q(3),
      S = J(3) + Q(3)'
```

RESULT: The data base variables R and S will contain the combinations indicated. Any of the arithmetic operators (+, -, *, /, **) may be employed, up to 10 operations may be performed.

ADDITIONAL RESTRICTIONS:

1. The element number cannot be a data base name.

2. Mixed mode arithmetic cannot be performed (i.e. integer * real). Unpredictable results will occur.

5.2.2 The DEFINE Command

The ADD command described in the previous paragraphs is the most useful in the communication command language. However, the DEFINE command described below will be useful for two purposes:

1. To reserve space in the data base for data bases variable before the data is actually entered.

2. To provide a brief description of the data base variables. This description is stored in the data base.

As with the ADD command, the DEFINE may be used anywhere within the execution sequence. However, it is most likely to be used in the creation of a design data base for reserving space or providing definitions for new or existing variables.

The format of the DEFINE command is:

'DEFINE name = n, description,'

The following SYNTAX or formation rules apply to the DEFINE command:

1. The opening delimiter (') may be any column.

2. No spaces may appear in the character string 'DEFINE.'

3. One or more spaces must appear between the DEFINE and the name.

4. The n is a number of data base locations to be reserved. If omitted one will be assumed. If previously defined n will be ignored.

5. A comma must separate the name = n set and the description.

6. The description may be up to 30 characters (see Section 4 to change this number).

7. The description must be terminated with a comma (,).

8. The DEFINE command must be terminated with a DIALOG delimiter (').

More than one variable may be defined by a single define statement. Any number of continuation cards may be employed. Although more than one variable may be defined on a single card, experiments have shown unpredicable results can occur from this practice. The usual technique is to define one variable per card and use many continuation cards. The most common SYNTAX errors result from failure to conform to rules 2, 5 and 8. The reader's attention is specifically drawn to these rules.

Figure 5-7 illustrates the use of the DEFINE command. They define the example variables used in Section 5.2.1. The definitions appear on the sample data base printout of Figure 5-6.

5.2.3 The Comment Command

The comment command is used in applications program data for identification only and as such, the comment performs no functional operation. The form of the comment:

```
'. comment'
```

The SYNTAX rules for construction of a comment command are:

1. The opening delimiters and closing delimiter may appear in any column.
2. No space may appear between the delimiter (') and the dot (.).
3. At least one space must appear between the dot (.) and the start of the comment.
4. The comment may be any number of characters.
5. The comment may appear on the same card with data.
6. The comment may not appear on the same card with another command (i.e. ADD or replacement command).
7. The comment may be continued on many cards so long as the comment does not start on the same card as data.
8. A comma (,) may not be used in a comment.
9. The comment must be terminated with a delimiter (').

The SYNTAX rules which are most often violated are 2, 3, 6 and 8. The attention of the reader is drawn specifically to these rules.

The DIALOG executive program replaces the comment and associated delimiters with blanks as they are encountered. After processing the comment, DIALOG checks to determine if the card is all blank. If blank, DIALOG "removes" the card from the input stream.

Examples of comments are given in both Figures 5-5 and 5-7.

```
≠. DEFINE STATEMENT EXAMPLES ≠

≠DEFINE A, REAL VALUE                              ,≠
≠DEFINE B, INTEGER VALUE                           ,≠
≠DEFINE C, HOLLERITH VALUE                         ,≠
≠DFFINE D, LOGICAL VALUE                           ,≠
≠DEFINE E, E=A                                     ,≠
≠DEFINE F, F=B                                     ,≠
≠DEFINE G, G=A*2.                                  ,≠
≠DEFINE H, H=B+3                                   ,≠
≠DFFINE I, I=A+G*2.                                ,≠
≠DEFINE J=3, REAL ARRAY                            ,≠
≠DEFINE K=3, INTEGER ARRAY                         ,≠
≠DEFINE L=3, HOLLERITH ARRAY                       ,≠
≠DFFINE M=3, MIXED LOG-INT-REAL ARRAY              ,≠
≠DEFINE N=4, SHORTHAND LOAD                        ,≠
≠DEFINE O=3, MIXED TYPE ARRAY                      ,≠
≠DFFINE P, P=J(1)                                  ,≠
≠DEFINE Q, Q=K(2)                                  ,≠
≠DEFINE R, R=A*O(3)                                ,≠
≠DEFINE S, S=J(3)+O(2)                             ,≠
```

NOTE: THE DELIMITER (≠) RESULTS FROM THE CHARACTER SET USED ON THE LISTING EQUIPMENT.

FIGURE 5-7 DEFINE COMMAND EXAMPLES.

5.2.4 Replacement Command.

Data base information is entered into the applications program input data by means of the replacement command. As the name implies the replacement command replaces delimited data base names with the corresponding values. Any delimited data base variable name may be placed on a data card of an applications program. The DIALOG executive program will replace the delimited name with the value (s) from the data base. Therefore, nearly any input procedure can be accommodated. The general format of the replacement command is:

'name'

The name is any data base name, combination of data base names and constants. Further, the names may be names of single variables or arrays. The results from the two types of replacement are different and will be treated by example. Figure 5-8 illustrates the use of the replacement command. The general rules governing the replacement are as follows:

1. The opening and closing delimiter (') may be placed in any column.

2. No space may appear between the opening delimiter and the first data base name.

3. Any combination of data base variables, array elements and constants may be used.

4. The first variable must be a name.

5. A maximum of 20 characters may be used between delimiters.

6. A maximum of 10 operations may be performed.

7. The number of significant places of the replaced number will be the maximum allowed by the space between (and including) the delimiters.

8. The opening and closing delimiter must be on the same card.

The most common SYNTAX errors result from failure to comply with rules 2, 4 and 5. Attention is specifically drawn to these rules. One additional problem may arise by not allowing enough space between delimiters to obtain the desired number of significant places.

'A'

In the above illustration, the number of significant places of the replacement value would be reduced to three places. The recommended replacement technique would be:

'A '

with sufficient space between delimiters to provide the desired significant places.

```
1          ABCS OF MODIFYING APPLICATIONS PROGRAM INPUT
0                    DATA BASE INFORMATION
0
0
0    NAME  COMBINATION     VALUE(S)
0    A                     ≠A    ≠
0    B                     ≠B    ≠
0    C                     ≠C    ≠
0    D                     ≠D    ≠
0    E     A               ≠A    ≠
0    F     B               ≠B    ≠
0    G     A*2.            ≠A*2.≠
0    H     B+3             ≠B+3  ≠
0    I     A+G*2.          ≠A+G*2.≠
0    J                     ≠J    ≠
0    K                     ≠K≠
0    L                     ≠L≠
0    M                     ≠M≠
0    N                     ≠N≠
0    O                     ≠O≠
0    P     J(1)            ≠J(1) ≠
0    Q     K(2)            ≠K(2) ≠
0    R     A*O(3)          ≠A*O(3)≠
0    S     J(3)+O(2)       ≠J(3)+O(2)≠
0          N*J             ≠N*J≠
0          J*N             ≠J*N≠
0          J*2.            ≠J*2.≠
0          J*A             ≠J*A≠
0          A*J             ≠A*J≠
0          A+J             ≠A+J≠
0          J+A             ≠J+A≠
0          A/J             ≠A/J≠
0          J/A             ≠J/A≠
1
```

NOTE: THE DIALOG DELIMITER RESULTS FROM THE CHARACTER SET USED ON THE CARD LISTING EQUIPMENT.

FIGURE 5-8 ILLUSTRATION OF REPLACEMENT COMMAND.

Figure 5-8 is hypothetical input stream, each line represents a card input. Delimited data base variables are placed on the card representing data base variables or combinations of data base variables, arrays and constants. The data base information comes from the data base of Figure 5-6. Figure 5-9 shows the results of the replacement commands in Figure 5-8. The following paragraphs discuss each of the examples with regard to objectives and results. The format of the descriptions follow the general format outlined below:

OBJECTIVE: A concise statement of the analysis goal to be achieved by the command.

SYNTAX: Pattern of formation of the command.

EXAMPLES: A list of one or more examples. Sometimes incorrect examples are given and are noted as ILLEGAL. In these cases the results are either incorrect or unpredictable for the objective stated.

RESULT: A brief description of the data base entry which will result from the use of the exemplified command.

ADDITIONAL RESTRICTIONS: In some cases, restrictions in addition to those described above will be given.

5.2.4.1 Simple Replacement of Data Base Names

OBJECTIVE: To replace a data base variable on an input card with a data base value.

SYNTAX: 'name'

EXAMPLES:

```
'A  '
'B  '
'C  '
'D  '
'K(2)'
```

RESULT: The delimited data base names will be replaced with the values in the data base as follows:

```
10.000
     2
ALPHA
.TRUE.
     4
```

DATA BASE INFORMATION

NAME	COMBINATION	VALUE(S)
A		10.000
B		2
C		ALPHA
D		.TRUE.
E	A	10.000
F	B	2
G	A*2.	20.000
H	B+3	5
I	A+G*2.	60.00000
J		10.000000000000000,20.000000000000000, 30.000000000000000,
K		2, 4, 6,
L		ALPHA ,BETA , GAMMA ,
M		.TRUE. , 3, 9.000000000000000,
N		8.000000000000000,8.000000000000000, 9.000000000000000,9.000000000000000,
O		2,5.000000000000000, 6.000000000000000,
P	J(1)	10.000000000000000,20.000000000000000, 30.000000000000000,
Q	K(2)	4

FIGURE 5-9 RESULTS OF THE REPLACEMENT COMMAND.

```
R     A*O(3)          60.00000

S     J(3)+O(2)       35.00000000

      N*J             80.00000000000000000,160.00000000000000000,
270.00000000000000000,

      J*N             80.00000000000000000,160.00000000000000000,
270.00000000000000000,

      J*2.            20.00000000000000000,40.00000000000000000,
60.00000000000000000,

      J*A             100.00000000000000000,200.00000000000000000,
300.00000000000000000,

      A*J             100.00000000000000000,200.00000000000000000,
300.00000000000000000,

      A+J             20.00000000000000000,30.00000000000000000,
40.00000000000000000,

      J+A             20.00000000000000000,30.00000000000000000,
40.00000000000000000,

      A/J             1.00000000000000000,.50000000000000000,
.33333333333333321491,

      J/A             1.00000000000000000,2.00000000000000000,
3.00000000000000000,
```

FIGURE 5-9 RESULTS OF THE REPLACEMENT COMMAND. (CONTINUED)

The values may be real integer, hollerith or logical. The maximum number of significant places defined by the delimiter will be used. Integers are right justified in the field. All others are left justified.

ADDITIONAL RESTRICTIONS:

1. The first element of an array cannot be used.

5.2.4.2 Simple Replacement of Data Base Combinations

OBJECTIVE: To replace a data base variable combination with the computed value.

SYNTAX: 'name1 + value1 * name2'

EXAMPLES:

'A * 2.'

'B + 3'

'A + G * 2.'

'A * O(3)'

'J(3) * O(2)'

RESULTS: The delimited data base variable or array element combination will be replaced with the computed values as follows:

20.000

5

60.000

60.00000

35.00000000

The arithmetic operations are performed in a serial manner (as on a hand calculator) from the left.

ADDITIONAL RESTRICTIONS:

1. The item adjacent to the opening delimiter must be a name.

2. Mixed mode arithmetic is not permitted (i.e. real * integer) unpredictable results will occur.

3. The first element of an array cannot be used.

5.2.4.3 Array Replacement by Name

OBJECTIVE: To replace an array name with an array of information from the data base.

SYNTAX: 'name' or 'name (1)'

EXAMPLES: 'J' or 'J(1)'

'K'

'L'

'M'

The arrays are placed on the card starting at the first delimiter using 20 characters per element, three elements per card. Elements are separated by commas. Continuation cards are created as required with data starting in column 2. The position of the closing delimiter is immaterial. 'J' or 'J(1)' produce the same result. Integers are right justified in the field, all others are left justified.

ADDITIONAL RESTRICTIONS:

1. Array replacement is generally limited to NAMELIST or special input procedures. It is probably not suitable for formatted input.

5.2.4.4 Array Replacement of Data Base Combinations

OBJECTIVE: To replace a data base combination on an element by element basis.

SYNTAX: 'name1 * name2'

EXAMPLES: 'N*J' (array * array)

'J*2.' (array * constant)

'J*A' (array * data base variable)

'A/J' (constant/array)

RESULTS: The operation is performed on an element by element basis. If one factor is a data base variable or constant, the one number is used as an operator on every element of the array. If both are arrays, the resulting array is equal in length to the shorter one. Division of a constant by an array results in an

element by element division of the constant by the elements of the array. For multiple operations, an element by element serial arithmetic is performed.

ADDITIONAL RESTRICTIONS: Reference to the first element of an array refers to the entire array.

A summary of the communication commands is given in Appendix E.

5.3 Standard Utility Procedures

The purpose of maintaining a program library for use with the DIALOG executive system is primarily for ready availability of applications programs. However, during the development and subsequent applications of the system, a set of utility programs or procedures has evolved for performing such tasks as:

a. Compilation and execution of interface programs.

b. Disposition of data files.

c. Report writing.

d. Executing arbitrary control card procedures.

Figure 5-10 summarizes the utility procedures available. These procedures are stored in the control card data base in exactly the same manner as applications program procedures. They are called into execution with the EXECUTE directive. The following paragraphs provide a brief description of the use of the standard utility procedures. Some have associated data and some do not.

5.3.1 COLOGO: Compile, Load and Execute a FORTRAN Program

Usually the submission of a job using the DIALOG executive system involves the execution of prestored applications programs. However, the need for an interface program is often desired or required in order to augment or transform the data obtained from existing applications programs. In these instances the analyst can generate a FORTRAN program within the execution sequence using the COLOGO procedure. This control card sequence compiles a FORTRAN program, then executes the compiled program. The execution of this procedure is implemented as follows:

'EXECUTE COLOGO'

FORTRAN Program

7-8-9

A brief summary of the rules of FORTRAN is given in Appendix F. It is not the intention of this appendix to provide a complete reference to FORTRAN but

COLOGO	Compile, Load and Execute
COMPILER	Compile User Program
MYPROGRAM	Execute Compiled Program
PRINTER	Prints Applications Program Data
REPORT	Prints Data Base Status Reports
ROUTECC	Routes Data to Central Site
ROUTEXP	Routes Status Reports to Terminal
CCSAVE	Permanent Update to DIALOG Executive System
PLOTSV	Saves Accumulated Plot
NEW PROC	Executes User Supplied Control Cards

FIGURE 5-10 DIALOG EXECUTIVE SYSTEM UTILITY PROGRAMS

rather to provide the user of the DIALOG executive system who has FORTRAN experience a ready reference to the FORTRAN statement formats. To avoid confusion, complex FORTRAN statements have been omitted. Only those FORTRAN features which are commonly used are presented. Detailed information on the use of FORTRAN is provided in standard FORTRAN reference manuals.

The use of COLOGO does not permit reading of data from cards. However, data base interfaces may appear in the source code such as:

```
A = 'DBA '
```

The above illustration is a FORTRAN statement which equates the FORTRAN variable A to the data base variable DBA. DIALOG preprocesses the entire FORTRAN program searching for delimited data base names and systematically replacing them with data base values. After processing, the illustrated statement might read:

```
A = 4.7562
```

In the illustration, the value of the data base variable DBA was 4.7562. The resulting statement is a legal FORTRAN statement. In effect the data base replacement commands are "input" to the FORTRAN program.

The use of the COLOGO procedure requires that the PROGRAM card for the user supplied FORTRAN program have a specified format.

```
PROGRAM MAIN (TAPE1, OUTPUT, TAPE78,---)
```

TAPE1 can be used for reading or writing a file of information for use within the simulation but no provision is made for reading data from cards. Other file parameters may be added beyond TAPE78. The OUTPUT file is the normal output file which can be printed with the PRINTER procedure (see Section 5.3.3). TAPE78 is the special output file for placing data base information. The file need not be TAPE78 but any convenient unused file specified by the user. This file can be a NAMELIST or simulated NAMELIST file as described in Section 4.

5.3.2 COMPILER/MYPROGRAM: Compile a FORTRAN Program/Execute the Compiled Program

Usually the COLOGO procedure provides all the FORTRAN capability needed by the analyst. However, COLOGO does not allow the reading of data from input cards. This capability is provided by combining the COMPILER and MYPROGRAM procedures in separate executions. COMPILER compiles the program while MYPROGRAM executes the compiled program. The data associated with COMPILER is the FORTRAN source code. The data associated with MYPROGRAM is the input data for the compiled program. The execution sequence is as follows:

```
'EXECUTE COMPILER'

    FORTRAN Source Code

7-8-9

.....

.....
```

'EXECUTE MYPROGRAM'

Data for Compiled Program

7-8-9

The use of the COMPILER/MYPROGRAM execution sequence requires that the PROGRAM card for the user supplied FORTRAN program have a specified format:

PROGRAM MAIN (INPUT, OUTPUT, TAPE5 = INPUT, TAPE6 = OUTPUT, TAPE78,....)

INPUT and OUTPUT are the normal read/write files. TAPE78 is the special data base output file as described in Section 4. Other file parameters may be added for special purposes but must be positioned beyond the TAPE78 file.

There are two advantages to using the above execution sequence rather than the COLOGO procedure:

a. Card input capability is available in the execution of MYPROGRAM.

b. The compilation can be physically separated from the execution.

The COMPILER/MYPROGRAM sequence may be useful for parametric studies and optimization problems where a single compilation can suffice for multiple executions of the same compiled programs. The use of the branching logic described in Section 5.1 is useful in this regard. Further, the COMPILER/MYPROGRAM procedures are useful for interface programs or testing new programs with the DIALOG executive system. These procedures together with COLOGO provide full FORTRAN capability as a subset of the DIALOG executive system.

5.3.3 PRINTER: Prints Output Generated in the Previous Execution

In the DIALOG executive system, the normal output from an applications program is stored temporarily on a file called OUT. This file is regenerated for each execution of an applications program. Therefore, the normal applications program output file is generally destroyed as a new applications program is being executed. However, the PRINTER procedure may be employed to alter the normal course of events by printing the OUT file. The format is:

'EXECUTE PRINTER'

7-8-9

The affect is to rewind the OUT file and copy it to OUTPUT. If the previous applications program or utility procedure generated output in the normal manner (WRITE (6,100)A), the PRINTER procedure will transfer the information to the output file.

5.3.4 PLOTSV: CALCOMP Plot Save Procedure

Many applications programs generate plotted information as a part of the normal calculations. The most common system for generating plotted information is the CALCOMP plotter system. CALCOMP consists of a set of FORTRAN callable software (subroutines) for generation of a file of plot commands. These plot commands are read from a physical tape by the CALCOMP hardware and are interpreted as pen movements on an x-y plot device.

For computational efficiency, submission of a plot generating program usually results in the generation of a disk file of plot commands called CALTPE. These commands must be transferred to a physical tape in order to be plotted.

The PLOTSV procedure mounts a physical tape and copies all CALCOMP plots generated previously in the execution sequence from disk to the mounted tape. The CALTPE file is dropped after it is copied to tape. Any number of executions of this procedure may be employed. A CALCOMP tape will be generated for each execution. The format is:

```
'EXECUTE PLOTSV'

7-8-9
```

5.3.5 REPORT: Generates Data Base Status Report

Usually the submission of a computation to the digital computer results in the generation of detailed information about the process involved. The results as well as intermediate information are printed by the normal output channels. The submission of the same computation using the DIALOG executive system involves the generation of the same information plus some summary type information which is usually a small subset of the total output of the applications program. The summary information is placed on the special output file discussed in Section 4 and the normal output is disposed in the manner illustrated in Section 5.3.3.

The information on the special output file may be placed in the data base after which it is generally available for printing through the REPORT procedure.

```
'EXECUTE REPORT'

    report data

7-8-9
```

The report data is a sequence of punched cards much like the input data to an applications program. However, the report data is not processed by any computer program but simply printed by the DIALOG executive program.

The report data is formatted by the analyst to provide any descriptive information desired. Further the report data may contain data base information through the

use of the communication commands described in Section 5.2. An example card in the report might be:

WEIGHT OF THE SYSTEM IS 'WGT' POUNDS

In the above illustration, WGT is a data base variable. The DIALOG executive program replaces the data base name and delimiter 'WGT' with the information stored in the data base. The report is printed after processing the report data. The result is a "stylized report" specifically tailored to the needs of the analyst. The report may contain "carriage control characters" in column 1 of the report data cards.

1 - eject a page before printing

0 - skip a line before printing

Any number of reports may be generated during a simulation. Usually during the initial phase of coordinating of large simulation using the DIALOG executive system, the staff selects subsets of the data base information to be communicated to each staff member for analysis. The format of the individual reports is tailored to the needs of the individual receiving the information. Once the format is established, it is keypunched on data cards with data base information being identified by name in the manner described in Section 5.2. These data cards become a report file.

A mini-report exemplifying this technique is shown in Figure 5-11. Any of the features of the DIALOG language including scaling and adding data base information are used in a completely free field report format. The first column of each card is reserved for printer carriage control providing a convenient means of paging and spacing for report clarity. Figure 5-11 also shows the printed results of the report file with data from the data base.

5.3.6 ROUTECC: Route Normal Output to the Central Site

This procedure was developed for use at any remote terminal. It routes the OUT file (containing the normal output from the previous execution sequence) to central computer facility. The output will be returned by the normal delivery service.

The ROUTECC procedure is quite useful for the disposition of voluminous data generated by some applications programs, when the analyst wishes to limit the printed output at a terminal. The format is:

'EXECUTE ROUTECC'

```
                                            ≠EXECUTE REPORT ≠
                                            ≠. DATA FOR SUMMARY REPORT ≠
1                                                                    PAGE 1

                        SUMMARY REPORT FOR ODIN/RLV
(≠ JJJ≠ CYCLE(S)                                  ELAPSED TIME = ≠ELTIME≠ CPU SECONDS
  ≠ADD A=TCOSTO+TCOSTB≠
  ≠ADD KN=1.689≠
  ≠ADD K=0.45359≠
 MAXIMUM PAYLOAD              ..... ≠WPAYLO  ≠ LBS          ≠WPAYLO*K≠ KILOGRAMS
        BOOSTER WEIGHT ..... ≠WGROSB  ≠ LBS          ≠WGROSB*K≠ KILOGRAMS
        ORBITER WEIGHT ..... ≠WGROSO  ≠ LBS          ≠WGROSO*K≠ KILOGRAMS
 TOTAL COST                   .....$≠A       ≠ MILLIONS
        BOOSTER COST          .....$≠TCOSTB  ≠ MILLIONS
        ORBITER COST          .....$≠TCOSTN  ≠ MILLIONS
 STAGING VELOCITY             ..... ≠VSTGB   ≠ FPS          ≠VSTGB/KN≠ KNOTS
 ENGINE VACUUM THRUST         ..... ≠THRUST  ≠ LBS
 BOOSTER MASS RATIO           ..... ≠ALPHA(2)≠
 ≠ADD N=JJBEST≠
 BEST CYCLE=≠JJBEST≠, THE ≠N*2-1≠TH RECORD ON FILE OUTSAV. N=≠N≠

                              END OF ODIN/RLV SUMMARY
                                                    REPORT
+
1
0000000000000000000000
```

```
                                                                   PAGE 1


                       SUMMARY REPORT FOR ODIN/RLV

    1 CYCLE(S)                                    ELAPSED TIME = 58.26500 CPU SECONDS

MAXIMUM PAYLOAD            ..... 31845.6096 LBS         14444.8500 KILOGRAMS
        BOOSTER WEIGHT ..... 2671117.97 LBS         1211592.40 KILOGRAMS
        ORBITER WEIGHT ..... 713186.621 LBS         323494.319 KILOGRAMS

TOTAL COST                 .....$67809.5935 MILLIONS
        BOOSTER COST       .....$48835.2395 MILLIONS
        ORBITER COST       .....$18974.3540 MILLIONS

STAGING VELOCITY           ..... 9202.29158 FPS ,       5448.36683 KNOTS

ENGINE VACUUM THRUST       ..... 525000.000 LBS

BOOSTER MASS RATIO         ..... 2.72500000

BEST CYCLE=         1, THE         1TH RECORD ON FILE OUTSAV. N=  1


                        END OF ODIN/RLV SUMMARY REPORT
```

FIGURE 5-11   ILLUSTRATION OF REPORT GENERATION CAPABILITY.

5.3.7 ROUTEXP: Dynamically Prints Report at the Originating Terminal

This procedure allows the analyst to print status reports (as described in Section 5.3.5) on the simulation dynamically while the job is executing. The user composes the status report in any desired format. Information may be drawn from the data base by delimiting data base names in the same manner as with applications program input.

```
'EXECUTE ROUTEXP'

    status report

7-8-9
```

Immediately upon execution of this procedure, the user supplied status report, as modified by data base inforamtion, is printed even though the simulation may still be executing.

5.3.8 CCSAVE: Permanent Storage of the DIALOG Executive System Including CCDATA

This procedure copies the DIALOG executive system to data cell using the REPLACE utility. The execution of the CCSAVE procedure results in a new data cell location for the system. The DIALOG executive system consists of a number of files which CCSAVE copies.

ODIN	Initialization procedure
DIALOG	Execution procedure
ABEND	Abnormal end procedure
PINIT	Post initialization procedure
ODIALOG	Absolute element program for DIALOG
ODBINIT	Absolute element program for DNINIT
CCDATA	Control card data base

These files are discussed in detail in Section 4.4.

The procedure is activated with the following two cards:

```
'EXECUTE CCSAVE'

7-8-9
```

5.3.9 NEWPROC: Execution of an Arbitrary Sequence of Control Cards

The NEWPROC procedure allows the analyst to execute an arbitrary sequence of control cards in the following manner:

```
'EXECUTE NEWPROC'

    sequence of

    control cards

    CCLINK, DIALOG

7-8-9
```

The affect of the NEWPROC procedure is to give job control to the operating system for a sequence of control card executions then to return control to the DIALOG executive system. In affect, the operating system utilities become a subset of the DIALOG executive system through the use of NEWPROC.

5.3.10 ENDODN: To Save a Design Data Base for Future Use

Usually the execution of the DIALOG executive system requires the creation of a design data base which is used and modified during the simulation process but not saved at the end of the run. It is sometimes desirable to save the design data base and restart the simulation at a later date. Examples of this requirement would be a very long simulation involving many program executions or an optimization involving repetitive evaluations through a series of application programs.

The ENDODN procedure allows the analyst to save the data base in the precise configuration required to restart the simulation. The control card sequence follows:

```
ENDODN =

COMMENT.

FETCH, A____, SPR____, BINARY,, DUM.

DROPFIL, DUM.

REPLACE, DUM, DBASE.
```

It is not essential (nor required) to execute this provedure using the EXECUTE directive. If the ENDODN procedure is stored in the control card data base, it will be executed at the proper time (after the END directive). The data cell location must be specified by the analyst and retrieval as part of the initialization procedure. The retrieval is discussed in Section 4.

5.4 Special Options in DIALOG Executive Program

Generally any name, except communication command names, may be used to identify data base information. However, there are a few additional exceptions. The excluded names are:

BUILD

DBDUMP

ELTIME

INDUMP

OUTDMP

PAGDMP

which represent special commands to DIALOG. The data base variables are stored with the ADD command (i.e. 'ADD BUILD). If present in the data base, DIALOG will perform special functions as described in the following paragraphs.

The BUILD option is associated with the dynamic construction of the design data base by the applications programs. Two values have significance to the DIALOG executive program.

BUILD = 0

BUILD = 1

The zero value specifies that previously undefined variables will not be defined by an applications program. A value of one specifies that all information from the previous program will be stored in the data base regardless of its previous data base status. The value of BUILD may be changed from program-to-program by use of the ADD command:

'ADD BUILD = n'

in the input data of the applications program. The change in the BUILD option becomes effective for the program where it occurs and remains effective until changed again.

The BUILD option only affects data coming from applications programs. It has no affect in ADD commands or other user communication commands.

DBDUMP is a DIALOG option which specifies that the entire data base be printed after each applications program execution. An example of this printed output is shown in Figure 3-11. It should be noted that this option is mandatory when selected. Selective printing of the data base is accomplished by the PRINT directive:

'PRINT DBASE'

The directive illustrated above is discussed in Section 3.1.

ELTIME is a timing option. It is selected by a setting ELTIME to an initial value:

'DEFINE ELTIME = 7, description,'

When selected, a timing routine in DIALOG will be activated. This routine monitors the individual and cumulative computer processing parameters for the applications programs. The parameters are identified on the printed output.

INDUMP is an optional data base name which specifies the printing of the modified input stream for the applications programs. The modified input stream represents the input file in exactly the form the applications program will read it. The INDUMP option is useful in the early phases of linking programs for debugging purposes.

OUTDMP is an optional data base name which specifies the printing of the special data base output file, NMLIST, which contains all the information available for entry into the data base. Details of the NMLIST file and how it is used are discussed in Section 3.2.4. The OUTDMP can be used to determine what information is available from a given applications program or simply as a debugging aid in the early phases of the analysis.

PAGDMP is an option available to the programmer for detecting errors within the DIALOG executive system. It has little use to the analyst using the system. It is mentioned only because PAGDMP is an excluded data base name which has special significance to the DIALOG executive system.

A summary of special options and additional restrictions is given in Appendix G.

6.0 APPLICATIONS

The DIALOG executive system was developed to provide the program linking capability required in the Optimal Design Integration (ODIN) procedure of Reference 9. Two applications were used to demonstrate the ODIN capability and are presented here to illustrate the use of the DIALOG executive system in linking independent programs. The ODIN application program library is shown in Figure 6-1. This library was used to perform the analysis presented in this section. The results indicate that the DIALOG executive system has unlimited potential in solving a wide variety of engineering problems.

6.1 Orbiter Landing Skin Temperature Study

An example of a small ODIN problem is shown in Figure 6-2. A study was required to determine if landing performance or stability and control might be effected by the presence of excessive skin temperatures on the Space Shuttle Orbiter. This problem was formulated in ODIN to determine skin temperature time histories using the various ODIN technology programs. The MINIVER (a mini-version of the MDAC JATO Aerodynamic Heating Program) and ABLATOR (Reference 11) programs required were quickly integrated in the ODIN system. MINIVER was used to obtain convective heat rates along the entry trajectory and ABLATOR enabled calculations of the associated skin temperature variations over the orbiter surface. Formulation of the problem required the combined efforts of a group of engineers with technological backgrounds in materials, flight dynamics and aerothermodynamics. The deck setup for the problem is shown in Figure 6-3. The problem was solved approximately one week after its conception. The results, shown in Figure 6-4, indicated that no excessive temperatures were present on the orbiter skin during approach and landing using either reusable or ablative material.

6.2 Shuttle Orbiter Wing Design Study

A somewhat more complex demonstration of the ODIN system is summarized in Figure 6-5 for a shuttle orbiter wing design study. The purpose of this study was to provide an orbiter wing which would achieve hypersonic/subsonic compatibility. Study guide lines were chosen in accordance with those from NASA's Shuttle Request for Proposal in February, 1972. The orbiter wing geometry was perturbed over a matrix which yielded design data for 37 possible configurations. The ODIN design sequence progressed downward in the figure for each wing design and ended with aerodynamic data plots, a configuration drawing and a summary report of geometric and aerodynamic performance data. The technique allowed the users to converge rapidly on a viable orbiter wing design which was subsequently proven with the aid of minimal wind tunnel development. Figure 6-6 illustrates the type of results from the orbiter wing design study.

ODIN APPLICATIONS PROGRAMS

AERODYNAMICS
- SKINF
- DATCOM
- HABACP
- TREND
- WDRAG

WEIGHTS
- WAATS
- VAMP

STRUCTURES
- SSAM
- AFSP

HEATING
- MINIVER
- ABLATOR

OPTIMIZATION
- AESOP

GEOMETRY
- CONFIGURATION
- WETTED
- PANEL

TRAJECTORY
- ATOP
- PRESTO
- POST

COST
- DAPCA
- PRICE

SIZING
- VSAC
- SSSP

PLOTTING
- PLOTTER
- IMAGE

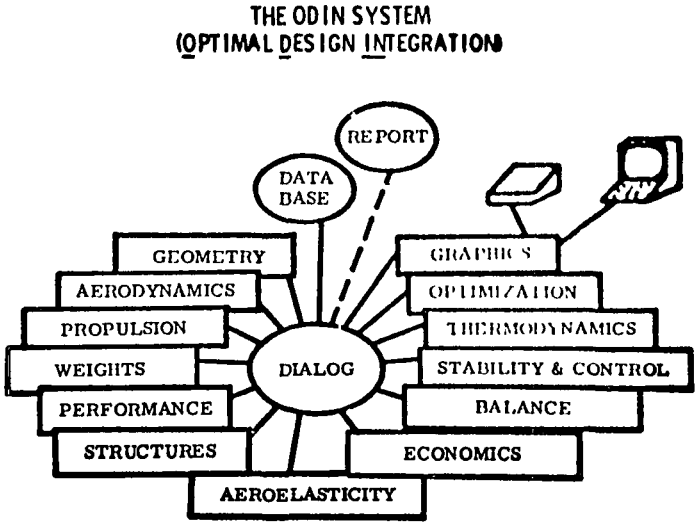
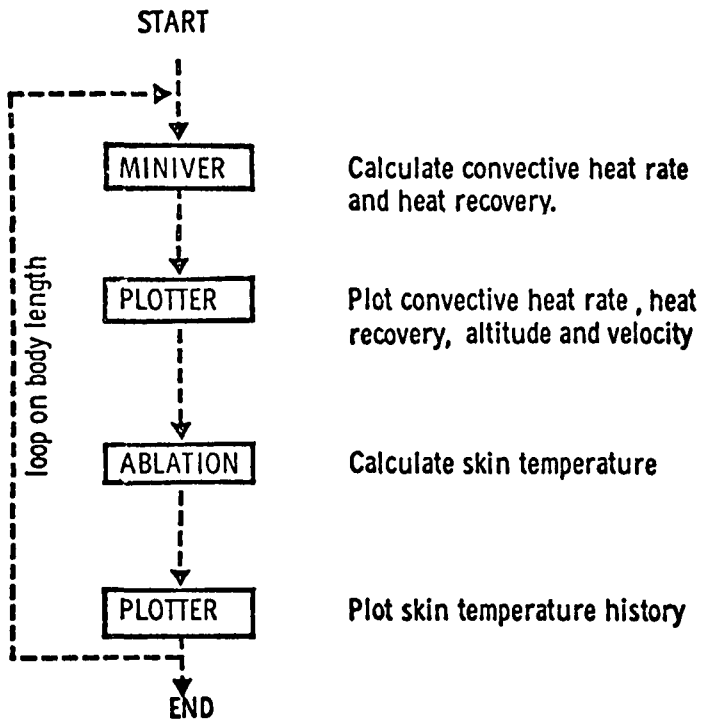


FIGURE 6-1 ODIN APPLICATIONS PROGRAMS



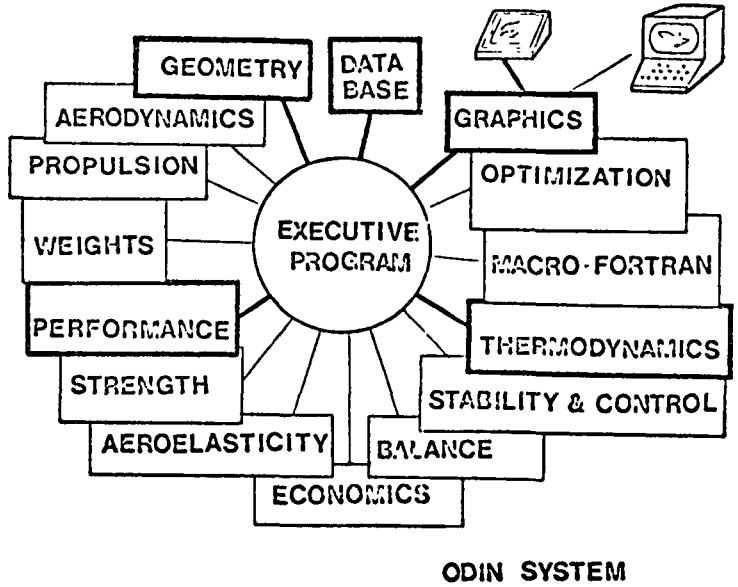


FIGURE 6-2 LANDING SKIN TEMPERATURE STUDY FOR ORBITER

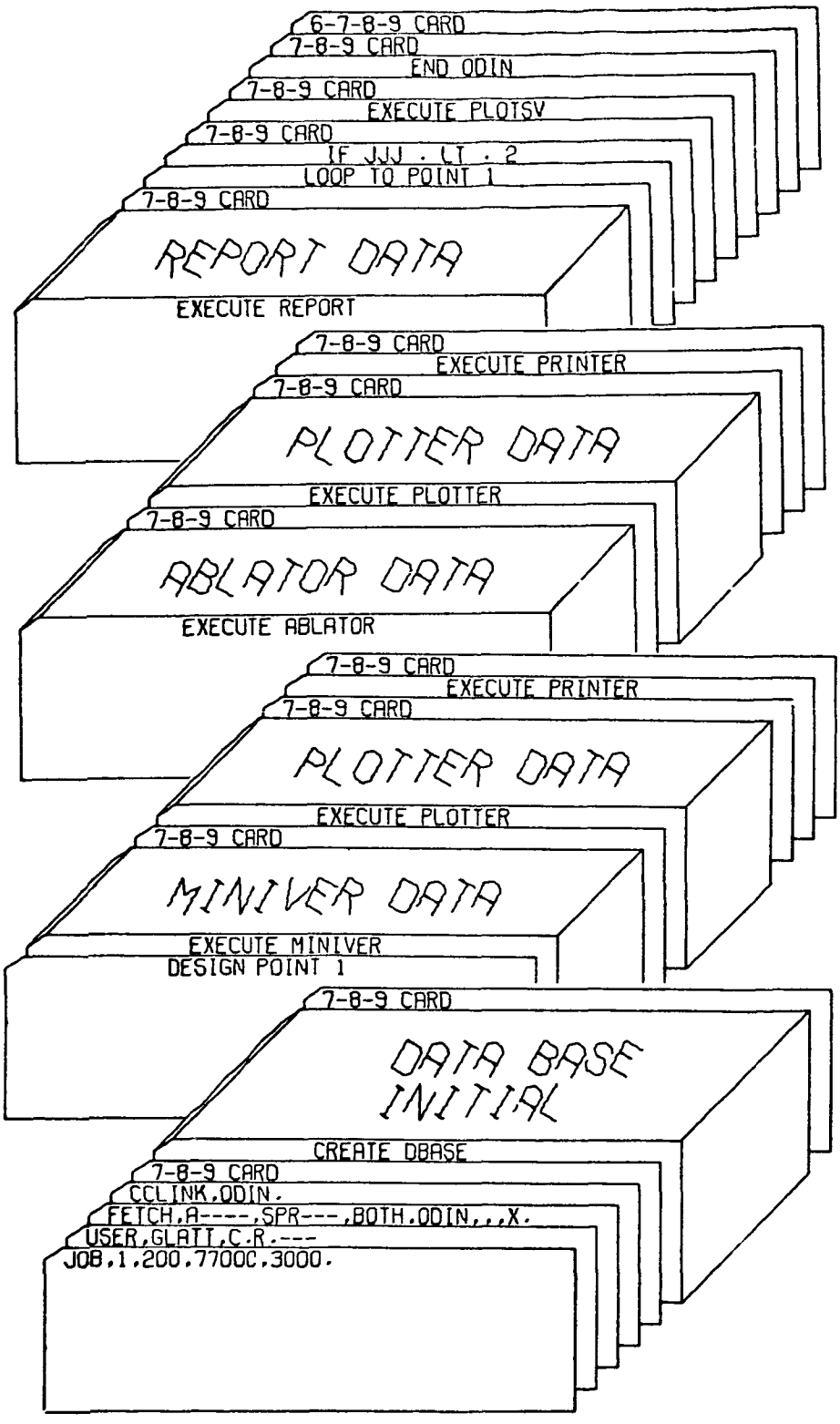


FIGURE 6-3 DECK SET UP FOR ORBITOR LANDING SKIN TEMPERATURE STUDY

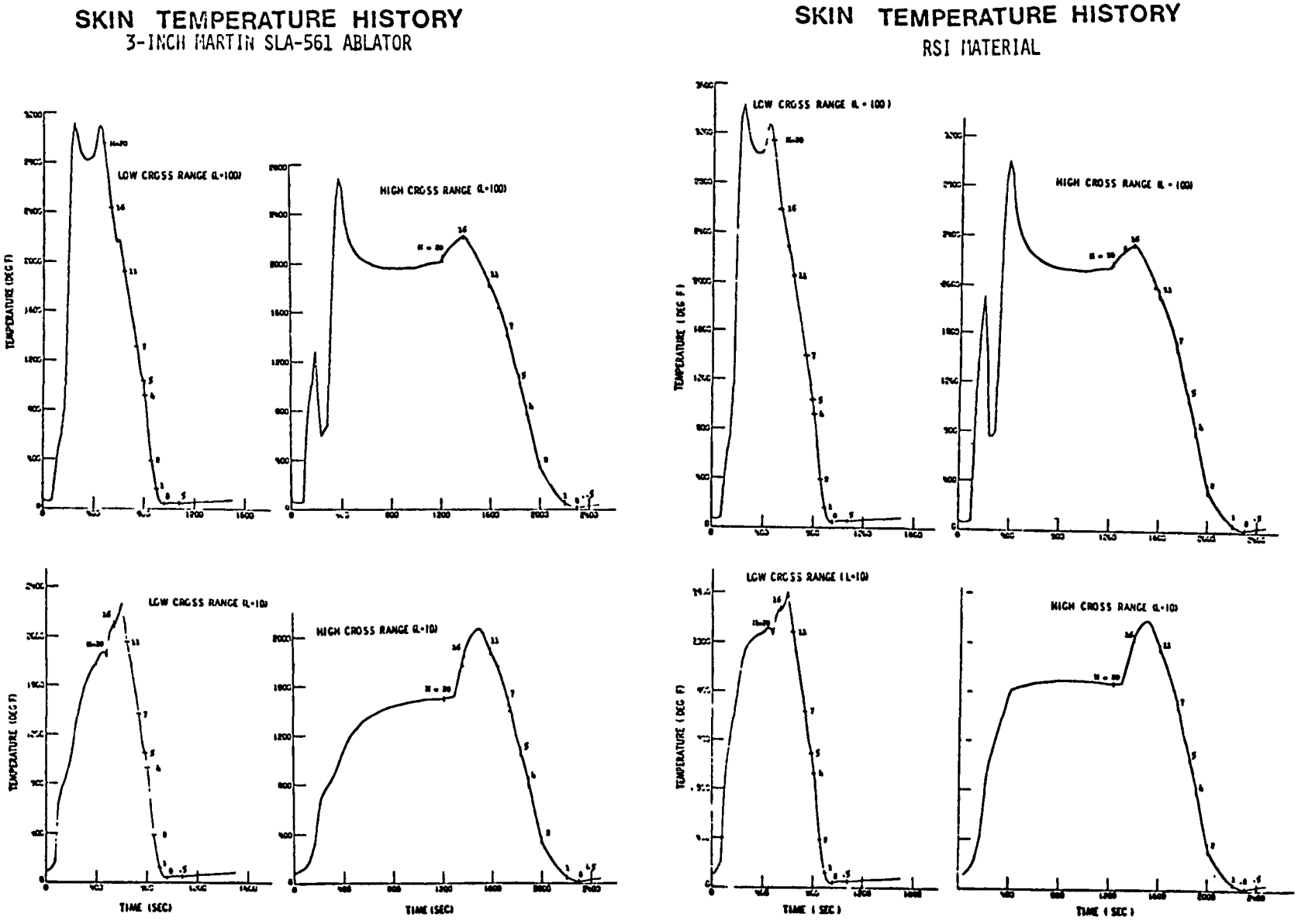
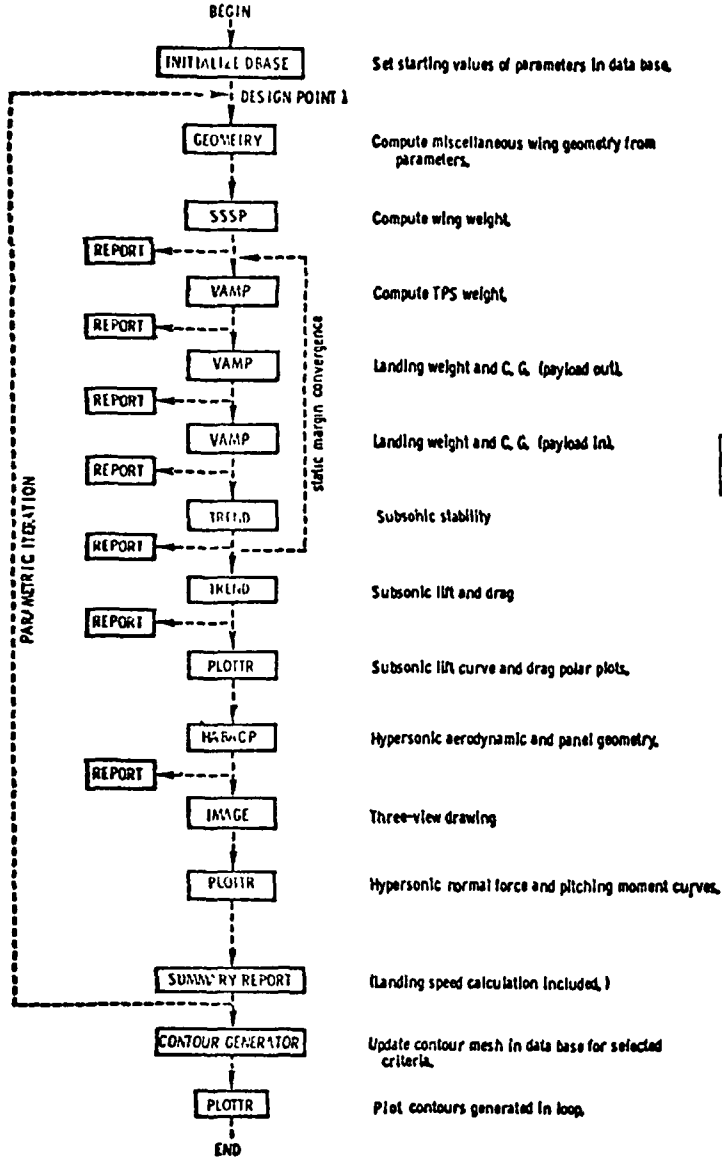


FIGURE 6-4 RESULTS OF THE ORBITER LANDING SKIN TEMPERATURE STUDY



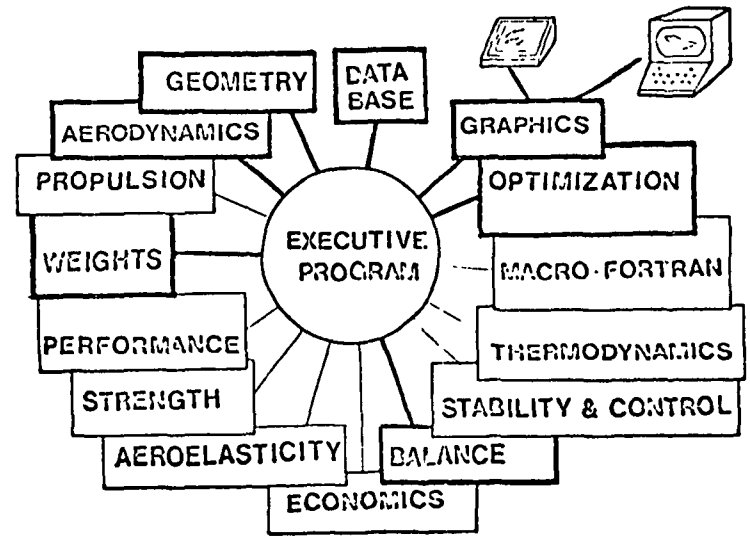


FIGURE 6-5 SHUTTLE ORBITER PITCH TRIM MISMATCH STUDY

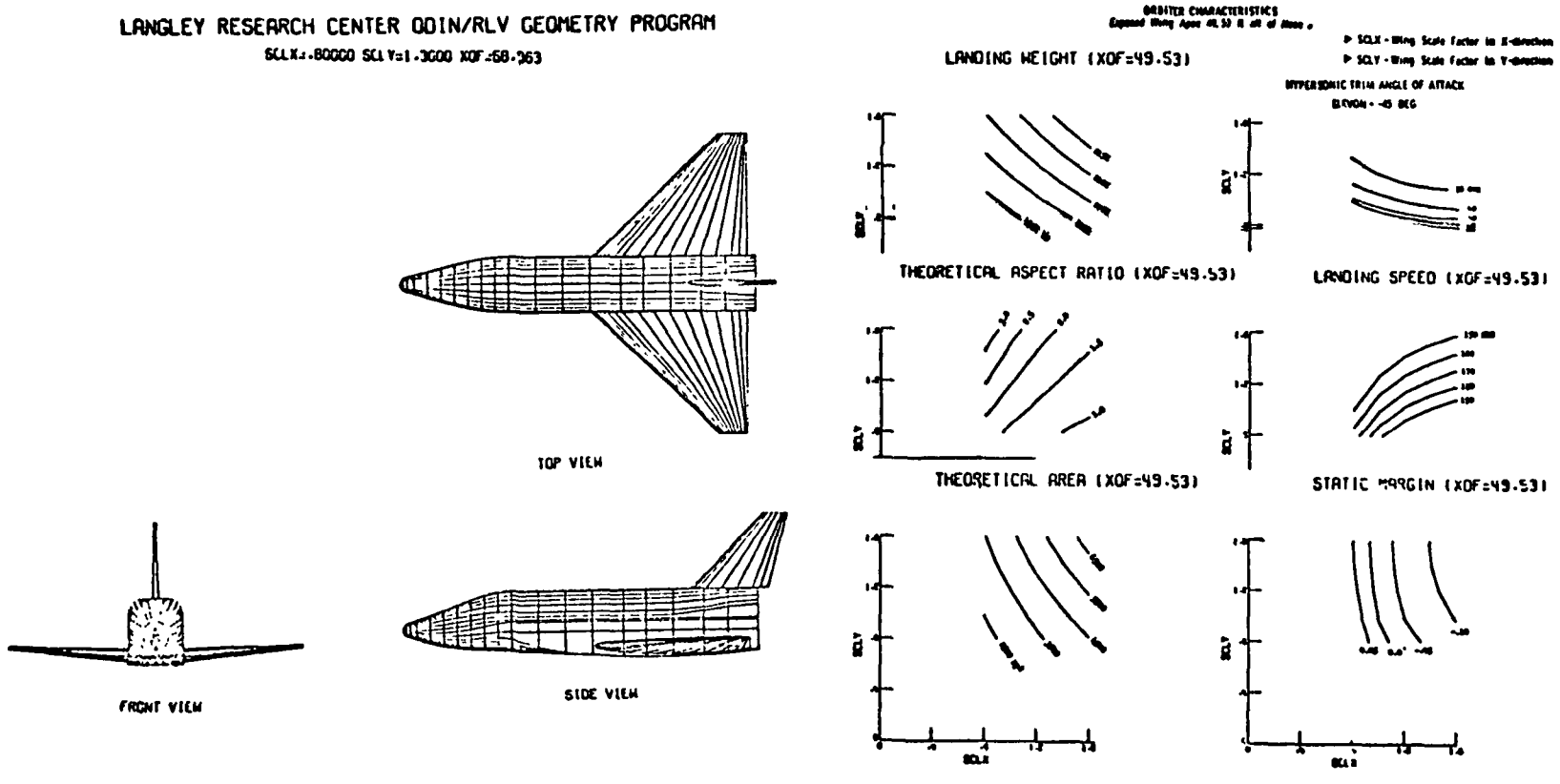


FIGURE 6-6 TYPICAL RESULTS FROM THE ORBITER PITCH TRIM MISMATCH STUDY

7.0 CONCLUSIONS

A very large scale synthesizing system for engineering processes has been described. The elements of the system are a library of independent applications computer programs, an executive computer program and a data base which forms the common information link among the independent applications program. The programs can be used by individuals for small problems or the operation can employ the design team approach. In the latter case, the design team defines the program sequences, data interfaces and matching loops required to achieve the desired design objective. The system provides the users with the ability to formulate the computer aided design problem at the task level in much the same manner as is employed in the industrial design process.

The executive computer program DIALOG controls the sequence of execution of the independent program elements and performs the data management function through the maintenance of the data base of common information. Each program is executed sequentially and as such is "unaware" of its contribution to a larger engineering process. DIALOG interrogates the data into and out of the independent programs and performs data manipulations according to "instructions" embedded in the data. The "instructions" are user supplied and form the control and communication language which are input to the DIALOG executive program. DIALOG restructures the input stream based on the instructions. The result is a flow of information which is not unlike the normal flow of individual jobs.

The greatest single advantage of the DIALOG executive system is that it allows full use of virtually all past developments in engineering technology for the synthesis of engineering processes. Any existing checked out computer code can be easily incorporated into the system library and the developer of new technological modules is unconstrained by requirements of the DIALOG executive system. Little or no programming knowledge is required to incorporate a program into the system for the first time.

The control and communication language consists of a simple and easily understood set of instructions which provide the capability of creating a network of computer programs for analysis at any level of detail. All synthesis processes and data intercommunication are performed at the program input level. Conditional branching logic is provided for creating sizing and/or optimization loops within the synthesis. There is no effective limit to either the number programs used or the complexity of design loops created.

The manual data transfer from technology to technology may be drastically reduced using the DIALOG executive system. Further the chance of data error, data misunderstanding or data misrepresentation is virtually eliminated. All factors dealing with "engineering judgment," "design margins" and "non-optimum analysis" may be employed and are visible to the design team.

Data visibility has been a key requirement in the development of the DIALOG executive system. Report generation is an integral part of DIALOG. User generated reports based on data base information can be generated at any point

in the sequence of program executions. A variety of graphical capability is available in the program library.

Finally the DIALOG executive system provides a true building block approach to the synthesis of engineering processes. Applications programs may be added, deleted or replaced to suit the design objective. This provides a responsiveness of computer aided design techniques never before available to the designer. All or any part of the design process may be synthesized but when using the DIALOG executive system, the designer never relinguishes his option to perform the analysis by alternate means, including hand calculation.

The software associated with the DIALOG executive system is written in the FORTRAN source language and is relatively machine independent. Machine dependent and system dependent code is used only when absolutely essential to the proper function of the DIALOG executive system. Where used the machine dependent code is isolated for quick conversion to other machines and other systems. Versions of the DIALOG executive system for CDC 6000 series and Univac 1100 series computers have been developed.

8.0 REFERENCES

1. Gregory, T. J.: Peterson, R. J.; and Wyss, J. A.: Performance Trade-Offs and Research Problems for Hypersonic Transports. AIAA Journal of Aircraft, July-August 1965.

2. Peterson, R. H., Gregory, T. J. and Smith C. L.: Some Comparisons of Turboramjet-Powered Hypersonic Aircraft for Cruise and Boost Missions. Journal of Aircraft, September-October, 1966.

3. Gold, R. and Ross, S.: Automated Mission Analysis Using a Parametric Sensitivity Executive Program. AAS Paper 68-146, presented at the AAS/AIAA Astronautics Specialist Conference, September 1968.

4. Wennegal, G. J.; Mason, P. W. and Rosenbaum, J.D.: IDEAS, Integrated Design and Analysis System. SAE Paper 68-0728, presented to SAE Aeronautics and Space Engineering Meeting, October 1968.

5. Adams, J. D.: Vehicle Synthesis of High Speed Aircraft, VSAC, Volume 1. USAF AFFDL-TR-71-40, 1971.

6. Oman, B.: Vehicle Synthesis for High Speed Aircraft, VSAC, Volume II. USAF AFFDL-TR-71-40, 1971.

7. Lee, V. A.; Ball, H. G.; Wadsworth, E. A.; Moran, W. J. and McLead, J.D.: Computerized Aircraft Synthesis. AIAA Journal of Aircraft, September-October 1967.

8. Herbst, W. B. and Ross, H.: Application of Computer Aided Design Programs for the Management of Fighter Development Projects. AIAA Paper 70-364, presented at the AIAA Fighter Aircraft Conference, March, 1970.

9. Hague, D. S. and Glatt, C. R.: Optimal Design Integration of Military Flight Vehicles - ODIN/MFV. AFFDL-TR-72-132, 1973.

10. Morris, Robert: Scatter Storage Techniques, Communications of the ACM. Volume II, No. I. January, 1968.

11. Swann, R. T., et al: One-Dimensional Numerical Analysis of the Transient Response of Thermal Protection Systems. NASA TN-D-2976, 1965.

APPENDIX A - CONTROL CARD SUMMARY

The use of the DIALOG executive system requires input of many forms to perform a complete analysis. The common forms of input include normal data to the applications programs but may include source language input or operating system control cards. This appendix provides a brief summary of control card functions at the Langley Research Center (LRC) Computer Complex.

Common requests that can be issued to the SCOPE operating system at Langley Research Center on control cards are noted in the following summary. Parameters issued as part of each request are briefly described. In this summary, constants are capitalized and variables are in lower case; the variables are defined below the illustrated format. In the formats shown, commas are used as separators and periods are used as terminators in all cases. Parentheses may be substituted for the opening comma and trailing period.

Following the definition of each control card, the function executed by that request is given. For more details of each control card, the reader is referred to the LRC computer reference manuals.

AUTOLAY,NEW,LIB1,LIB2,LIB3,LIB4,LIB4,LIB6. *

Combines library subroutines into a new program file.

NEW = new program file

LIB1 - LIB6 = Up to 6 library files

AUTOLAY is not a system control card; therefore, the program must be retrieved from storage as follows:

```
FETCH,A3596,SPRA04,BINARY,,AUTOLAY.
```

*This control card is described in more detail in Section 4.

CCLINK,lfn,xx,n

lfn = the logical file name of the linkage file

xx = a conditional operator (one of the following)
- LT (less than) link if CCIR LT n (See SETIDEX)
- LE (less, equal)
- GT (greater than)
- GE (greater, equal)
- EQ (equal)
- NE (not equal)
- omitted (unconditional linkage implied)

n = the comparison integer

COMMENT.n...n

Inserts comments in the day file.

n = comment characters

COPY,lfn1,lfn2.

Copies all files from lfn1 to lfn2

lfn = logical file name

COPYBF,lfn1,lfn2,n.

Copies n binary files from lfn1 to lfn2

lfn = logical file name

n = number of files (decimal)

COPYBR,lfn1,lfn2,n.

Copies n binary records from lfn1 to lfn2

lfn = logical file name

n = number of records (decimaCOPYCF,lfn1,lfn2,n.

Copies n hollerith or external BCD files from lfn1 to lfn2

lfn = logical file name

n = number of files (decimal)

COPYCR,lfn1,lfn2,n.

Copies n hollerith or external BCD records from lfn1 to lfn2

lfn = logical file name

n = number of records (decimal)

COPYSBF,lfn1,lfn2.

Copies lfn1 to lfn2 formatring binary file for single space printing

lfn1 = input file

lfn2 = output file

DROPFIL,lfn1,lfn2,...,lfnn.

Releases files and associated devices from job and decrements tape unit required count.

lfn = logical file names

EXECUTE,lfn,p1,p2,...pn.

Completes loading and linking of elements for execution, then executes this program

lfn = name of file containing program

p = parameters passed to program

EXIT.

Establishes exit path in event of selected errors.

FETCH,FILEN,DCNO,TYPE,SCFILE,BNFILE,DAFILE,XXX,Y.

To fetch a data cell file from the data cell to disk, thus making it available for use.

FILEN = name of data cell file, taken from label and printed in dayfile when program stashed or replaced.

DCNO = data cell name on which FILEN was written, given in dayfile when program stashed or replaced. If the word WEDGE is given for this parameter, FETCH will read from file TAPE95 to obtain the wedge name.

TYPE = SOURCE - Fetch only the source of FILEN

BINARY - Fetch only the binary of FILEN

BOTH - Fetch both source and binary

BINSEL - Fetch only selected BINARY routines

BOTHSEL- Fetch the source and selected BINARY routines

DATA - Fetch a BINARY data file

DATACRD- Fetch a BCD data file

SCFILE = disk file name on which FETCH stores source file, Default = SCFILE

BNFILE = disk file name on which FETCH stores BINARY routines, Default = BNFILE

DAFILE = desk file name on which FETCH stores data file, Default = DAFILE

XXX = NOCOM = control word which tells FETCH whether to read this file from the data cell or to use the COMMON file

Y = no MOD parameter - if TYPE file is fetched which expects mods from INPUT, but no modifications are to be made, any value may be put in this parameter instead of a 7-8-9 card in the INPUT deck.

The INPUT file is the 12th file parameter.

lfn,p1,p2,...pn.

Loads and executes program

lfn = name of file containing program

p = parameters passed to program

LINECNT,n.

OUTPUT line count limit card

n = DECIMAL number of lines to limit the job. Copied files are not counted in the line count.

LOAD,lfn.

Loads program on lfn into central memory.

lfn = logical file name

MAP.

Core map produced by the loader.

MODEn.

Defines halt conditions.

n - type of halt

NOGO.

Directs loader not to execute loaded program.

NOMAP.

NO MAP is produced by the loader.

NORFL.

Directs loader not be reduce field length after loading.

REPLACE,SCFILE,LGO,LIST,CATALOG.

To replace the present version of a file on a data cell with an updated version.

SCFILE = disk file from which REPLACE picks up source routine to put on data cell, Default = SCFILE.

LGO = disk file from which REPLACE picks up BINARY version of SCFILE to be put on data cell, Default = LGO.

LIST = if this parameter is LIST or left blank, a new sequenced listing of the replaced source program is printed, if any other characters are put in this field, a listing is not produced.

CATALOG = SCAT, a catalog of the BINARY (without listing COMMON) is produced.

LCAT, a catalog of the BINARY, including the name and length of each COMMON variable in each routine, is produced.

For DATA:

SCFILE = disk file from which REPLACE picks up the new data file to be stored on data cell. There is no default file. This parameter is used for both types, DATA and DATACRD.

LGO and
CATALOG = irrelevant to data

LIST = irrelevant for type data, has same meaning as for programs if type DATACRD.

The sixth parameter is the INPUT file. The eighth parameter is the OUTPUT file.

REQUEST,lfn,DEN,X. REEL,RWY,USER,LABEL,NO.

Magnetic tape request card.

lfn = logical file name

DEN = density LO = 200 BPI
HI = 556 BPI
HY = 800 BPI

X = designates external (stranger) tape

COLUMNS 25-70

REEL = actual real number

SAVTP
CALTP
CALSV
GERTP
DDITP
SCRATCH

RW = RI = ring in (write)

RO = ring out (read only)

Y = S = short (approximately 200 ft.)

M = medium (approximately 1100 ft.)

L = long (approximately 2300 ft.)

USER = user initial (must be three characters)

LABEL = up to 20 character description

NO = employee number (for save tapes only)

REWIND,lfn1,lfn2,...lfnn.

Rewinds files named.

lfn = logical file name

ROUTE,lfn,0,n,4,0.

Prints at specified location.

lfn = logical file name

n = 1 = central site

25 = originating terminal

RUN,cm,f1,bl,if,of,rf.

Compiles FORTRAN source.

cm = G = compile, load and execute.

S = compile with source list.

L = compile with source and object list.

fl = object program field length. (not applicable)

bl = buffer lengths (2022 base 8). (not applicable)

if = compiler input file (INPUT).

of = compiler output file (OUTPUT).

rf = relocatable binary file (LGO).

SETCORE.

Sets core to zero at load time.

SETIDEX,n.

Increments the CCIR index register for use by CCLINK.

n = CCIR increment.

SKIPFF,lfn,n.

Skips file forward by n logical files.

lfn = logical file name.

n - number of files (decimal).

STASH,SCFILE,LABEL,BNFILE.

Stores information on data cell for the first time.

SCFILE = file containing source code.

LABEL = file containing the label information.

BNFILE = file containing binary information.

SWITCH,n.

Sets switch to on or off

n = sense switch number

UPDATE,ident=lfn1,...identn=lfnn.

Calls UPDATE to perform program library maintenance (directive record required).

ident = type of file.

lfn = logical file name.

list = optional parameters specify update modes, comments, rewind, format, etc.

UNLOAD,lfn1,lfn2,....lfnn.

Same as DROPFIL but does not decrement tape unit count.

lfn - logical file names.

APPENDIX B CONTROL CARD DATA BASE

```
* * * * * THERE ARE CURRENTLY    28 DATA BASE ENTRIES * * * * * * * * * * * * * * *
          ALPHABETIZED LIST OF LAST 100 ENTRIES FOLLOWS

NAME        LOCATION DIMENSION   CURRENT VALUE(S)     ORIGIN    DESCRIPTION

ABLATOR            1          6  FETCH(A3747,SPRZ10,BINARY,,OBLATR)

                                  OBLATR(MODIN,OUT,,,,NMLIST)
                                  DROPFIL,TAPE9.
                                  CCLINK(DIALOG)
                                  EXIT.
                                  CCLINK(ABEND)
AESOP             49          6  FETCH(A3589,SPRA02,BINARY,,OAESOP)

                                  OAESOP(MODIN,OUT,,,,,LUSOP,NMLIST)
                                  DROPFIL,TAPE1,TAPE2.
                                  CCLINK(DIALOG)
                                  EXIT.
                                  CCLINK(ABEND)
ATOP              97          7  FETCH(A3595,SPRZ11,BINARY,,ATOPIII)

                                  ATOPIII(MODIN,OUT,,,,,,,,,,,,,,NMLIST)
                                  DROPFIL,TAPE1,TAPE2,TAPE3,TAPE4,TAPE10,TAPE11,TAPE12,TAPE13,TA
                                  DROPFIL,TAPE15,TAPF16.
                                  CCLINK(DIALOG)
                                  EXIT.
                                  CCLINK(ABEND)
```

```
CCSAVE          153      18  COMMENT. SAVE ODIN SYSTEM

                             FETCH,A3972,SPRA07,SOURCE,RTEXT,,,,,,,,MODIN.
                             REWIND,ODBINIT,ODIALOG,CCDATA,DIALOG.
                             REWIND,ABEND,PINIT,BNFILE.
                             COPYBR,DIALOG,BNFILE.
                             COPYBR,ABEND,BNFILE.
                             COPYBR,PINIT,BNFILE.
                             COPYBR,ODBINIT,BNFILE.
                             COPYBR,ODIALOG,BNFILE.
                             COPYBR,CCDATA,BNFILE.
                             REWIND,ODIALOG,CCDATA,DIALOG,ABEND.
                             REWIND,ODIN,BNFILE,ODINSA1,RTEXT,ODINSA2.
                             REPLACE,ODIN,BNFILE,NULL,,ODINSA1,RTEXT,,OUT,ODINSA2.
                             DROPFIL,BNFILE,NULL,ODINSA1,RTEXT,ODINSA2.
                             DROPFIL,ODBINIT,PINIT,ODIN.
                             CCLINK,DIALOG.
                             EXIT.
                             CCLINK,ABEND.
COLOGO          297       8  COMMENT. COMPILE LOAD AND GO

                             REWIND,NEW.
                             RUN(S,,,MODIN,OUT,NEW.
                             NEW,USER1,OUT,NMLIST.
                             DROPFIL,NEW.
                             CCLINK,DIALOG.
                             EXIT.
                             CCLINK,ABEND.
COMPILER        361       6  COMMENT. COMPILE  A NEW PROGRAM

                             REWIND(NEWPGM)
                             RUN(S,,,MODIN,OUT,NEWPGM)
                             CCLINK(DIALOG)
                             EXIT.
                             CCLINK(ABEND)
```

```
DAPCA          409        5  FETCH(A3593,SPRZ10,BINARY,,ODAPCA)

                             ODAPCA(MODIN,OUT,,,NMLIST)
                             CCLINK(DIALOG)
                             EXIT.
                             CCLINK(ABEND)
DATCOM         449        6  FETCH,A3908,SPRZ10,BINARY,,DATCOM.

                             DATCOM,MODIN,OUT.
                             DROPFIL,TAPE1.
                             CCLINK,DIALOG.
                             EXIT.
                             CCLINK,ABEND.
HABACP         497        6  FETCH(A3634,SPRZ10,BINARY,,CCT)

                             CCT(MODIN,OUT,,,,,,ELEMT,BNDATA,,,,NMLIST)
                             DROPFIL,TAPE1,TAPE3,TAPE4,TAPE9,TAPE10,TAPE11.
                             CCLINK(DIALOG)
                             EXIT.
                             CCLINK(ABEND)
IMAGE          545        6  FETCH(A3647,SPRZ09,BINARY,,OIMAGE)

                             OIMAGE(MODIN,OUT,,,,ELEMT,,NMLIST)
                             DROPFIL,TAPE3,TAPE20.
                             CCLINK(DIALOG)
                             EXIT.
                             CCLINK(ABEND)
MINIVER        593        6  FETCH(A3748,SPRA07,BINARY,,OMINIVR)

                             OMINIVR(OUT,,MODIN,,,,,,,NMLIST)
                             DROPFIL,TAPE1,TAPE3,TAPE4,FILMPL.
                             CCLINK(DIALOG)
                             EXIT.
                             CCLINK(ABEND)
```

```
MYPROGRAM       641         6  COMMENT. EXECUTES NEW PROGRAM

                               SETCORE.
                               NEWPGM,MODIN,OUT,,,NMLIST.
                               CCLINK(DIALOG)
                               EXIT.
                               CCLINK(ABEND)
NEWPROC         689         4  COMMENT. MODIFY AND EXECUTE CONTROL CARDS

                               CCLINK(MODIN)
                               EXIT.
                               CCLINK,ABEND.
PLOTSV          721         9  COMMENT. PLOT SAVE PROCEDURE

                               REQUEST,TAPE98,HI,X.      CALTP,RIL,TRR,ODIN/RLV PLOTS,00072785
                               REWIND(CALTPE,TAPE98)
                               COPYBF(CALTPE,TAPE98)
                               UNLOAD(TAPE98)
                               DROPFIL(CALTPE)
                               REWIND(NULL)
                               COPYBF(NULL,NMLIST)
                               CCLINK(DIALOG)
PLOTTER         793         7  FETCH(A3680,SPRZ10,BINARY,,OPLOT)

                               REWIND(BNDATA)
                               OPLOT(MODIN,OUT,,,BNDATA,,NMLIST)
                               DROPFIL(TAPE2)
                               CCLINK(DIALOG)
                               EXIT.
                               CCLINK(ABEND)
POSTDATA        913         5  FETCH,A3993,SPRZ09,BINARY,,OPDAT.

                               OPDAT,MODIN,OUT,POSTAPE.
                               CCLINK,DIALOG.
                               EXIT.
                               CCLINK,ABEND.
```

```
POSTPLOT        953         6  FETCH,A3966,SPRA11,BINARY,,OPLT.

                               REWIND,POSTAPE.
                               OPLT,MODIN,OUT,,,POSTAPE.
                               DROPFIL,TAPE99,NMLIST.
                               CCLINK,DIALOG.
                               EXIT.
POST            849         8  FETCH,A3975,SPRZ12,BINARY,,POST.

                               REWIND,POSTAPE.
                               POST,MODIN,OUT,,,,,,,POSTAPE.
                               DROPFIL,TAPE1,TAPE2,TAPE10,TAPE11,TAPE13,TAPE12,TAPE14,TAPE15,
                               CCLINK,DIALOG.
                               EXIT.
                               CCLINK,ABEND.
                               CCLINK,ABEND.
PRICE           1001        5  FETCH(A3592,SPRZ04,BINARY,,OPRICE).

                               OPRICE(MODIN,OUT,,,NMLIST)
                               CCLINK(DIALOG)
                               EXIT.
                               CCLINK(ABEND)
PRINTER         1041        4  COMMENT. PRINTS NORMAL OUTPUT FROM PREVIOUS PROGRAM

                               COPY,OUT,OUTPUT.
                               DROPFIL,NMLIST.
                               CCLINK,DIALOG.
ROUTECC         1073        5  COMMENT.  ROUTES NORMAL OUTPUT TO CENTRAL SITE

                               ROUTE,OUT,0,1,4,0.
                               DETACH,OUT.
                               DROPFIL,NMLIST.
                               CCLINK,DIALOG.
ROUTEXP         1113        7  COMMENT. DYNAMICALLY PRINTS REPORT AT TERMINAL

                               COPYBF,MODIN,OUT.
                               REWIND,OUT.
                               ROUTE,OUT,0,25,4,0.
                               DETACH,OUT.
                               DROPFIL,NMLIST.
                               CCLINK,DIALOG.
```

```
SSSP        1169      5  FETCH(A3681,SPRA10,BINARY,,TTSS)

                         TTSS(MODIN,OUT,,,,,NMLIST)
                         CCLINK(DIALOG)
                         EXIT.
                         CCLINK(ABEND)
TREND       1209      5  FETCH(A3590,SPRA09,BINARY,,OTREND)

                         OTREND(MODIN,OUT,,,NMLIST)
                         CCLINK(DIALOG)
                         FXIT.
                         CCLINK(ABEND)
VAMP        1249      6  FETCH,A3683,SPRZ01,BINARY,,OVAMP2.

                         OVAMP2,MODIN,OUT.
                         DROPFIL(TAPE8,TAPE9,TAPE10)
                         CCLINK(DIALOG)
                         EXIT.
                         CCLINK(ABEND)
VSAC        1297      5  FETCH,A3969,SPRZ08,BINARY,,EMS.

                         FMS,MODIN,OUT.
                         CCLINK,DIALOG.
                         EXIT.
                         CCLINK,ABEND.
WAATS       1337      5  FETCH,A3983,SPRA02,BINARY,,OWATS.

                         OWATS,MODIN,OUT.
                         CCLINK,DIALOG.
                         EXIT.
                         CCLINK,ABEND.
WETTED      1377      6  FETCH,A3915,SPRZ08,BINARY,,CRC.

                         CBC,MODIN,OUT,NMLIST.
                         DROPFIL,TAPE9,TAPE12.
                         CCLINK,DIALOG.
                         EXIT.
                         CCLINK,ABEND.
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

APPENDIX C SPECIAL PROCEDURES FOR THE DIALOG EXECUTIVE SYSTEM

INITIALIZATION

```
COMMENT. ODIN / ATS SYSTEM  JAN 4, 1973
NOMAP.
NOREL.
REWIND(DCNS,ZOUNDS)
COPYBF(DCNS,ODINSA1)
COPYBF(ZOUNDS,ODINSA2)
COPYBR,BNFILE,DIALOG.
COPYBR,BNFILE,ABEND.
COPYBR,BNFILE,PINIT.
COPYBR,BNFILE,ODBINIT.
COPYBR,BNFILE,ODIALOG.
COPYBR,BNFILE,CCDATA.
REWIND,ODBINIT,ODIALOG,CCDATA,DIALOG,ABEND,PINIT.
ODBINIT,,OUT.
ODIALOG,,OUT,,,COPY5.
CCLINK,NMLIST.
EXIT.
REWIND,OUT.
COPYBF,OUT,OUTPUT.
```

DIALOG EXECUTION SEQUENCE

```
REWIND,DIALOG,OUT.
ODIALOG,COPY5.
COPYBF,CONTROL,NULL.
CCLINK,CONTROL.
```

ABNORMAL END PROCEDURE

```
COMMENT. ABNORMAL END PROCEDURE
REWIND,OUT.
COPYBF,OUT,OUTPUT.
REQUEST,TAPE98,HI,X.      CALTP,RIL,TRR,ODIN/RLV
REWIND,TAPE98,CALTPE.
COPYBF,CALTPE,TAPE98.
UNLOAD,TAPE98.
```

POST INITIALIZATION PROCEDURE

```
DROPFIL,LFNPERM,DCNO,DAFILE,SCFILE.
DROPFIL,NOCONOC,HYEELSL,ZOOKS,ZOUNDS,TAPE95.
CCLINK,CONTROL.
```

APPENDIX D CONTROL DIRECTIVE SUMMARY

* 'EXECUTE name'	A directive for executing a sequence of control cards by name. Any name for which a prestored set of control cards has been defined is legal.
* 'CREATE name'	File handling directive for initializing files; the two acceptable names are: DBASE - design data base CCDATA - control card data base
* 'UPDATE name'	File handling directive for updating files. The two acceptable names are: DBASE - design data base CCDATA - control card data base
'DESIGN name'	Control directive defining a point in the execution sequence for which control may be returned. The name cannot be the same as a data base variable.
'LOOP TO name'	Branching instruction referring to a design name. It can be conditional or unconditional.
'IF name.OP. name'	Condition for branching. Any number of conditions may be specified on separate cards after a LOOP directive. If more than one condition is specified, the logical .OR. is implied. That is, any one of the conditions satisfied will trigger the branch instruction.
'RESTRT'	Means use the existing data base. It must have previously been defined and stored.
'PRINT name'	File handling directive for printing files DBASE and CCDATE are optional names.

OP is a conditional operator (LT,LE,EQ,GE,GT)

* Data is expected; end of record (789) is required.

APPENDIX E COMMUNICATION COMMAND SUMMARY

'ADD A = B, ...'	Used to create a new data base entry or alter the information associated with an existing data base entry. A is a new or existing data base entry, scalar or vector. B is the update information which can be real, integer or logical constants, variables or scaled combinations of scalar or vector elements. Multiple commands can be executed with a single ADD statement.
'DEFINE A = n, description,'	Used to define new or existing entries in the data base. A is the new or existing data base entry. n is the desired number of data base locations. It is ignored if the entry exists, the default is 1 if omitted. description is the hollerith information associated with the variable A.
'.comment'	Used for placing descriptive information in the data stream. DIALOG replaces the comment and associate delimiters with blanks in the applications program data deck.

'A '	Used to replace data base names and delimiters on an input card with data base information. A may be a scaler or vector data base entry of real, integer, hollerith or logical type. A may be a combination of real or integer data base variables, array elements or constants.

APPENDIX F - FORTRAN STATEMENT SUMMARY

This appendix provides a summary of statements in the FORTRAN scientific language for programming **under** control of the SCOPE system on Control Data 6400/6500/ 6600 computers.

The information presented is not intended to be complete. The objective is to provide the user of the DIALOG executive system with a ready reference to the construction or syntax of commonly used FORTRAN features. As such, no definition or descriptive information is given about individual statements.

The information presented should be useful to those analysts who make use of FORTRAN through the use of the COLOGO and the COMPILER/MYPROGRAM procedures in the DIALOG executive system.

FORTRAN CODING FORM

Each line of a FORTRAN coding form represents the four fields of a punched card.

Field	Columns
Statement Number	1-5
Continuation	6
Statement	7-72
Identification	73-80

Statements may be identified by an integer from 1 through 99999. If a C, *, or $ appears in column 1, the remainder of the card is ignored by the compiler, but printed with the source listing as a comment.

A punch other than zero in column 6 identifies a card as a continuation of the statement from the preceding card.

A statement is written in columns 7 through 72, blanks are ignored.

Columns 73-80 may contain identification and serial numbers. These columns are ignored, but printed with the program listing.

The entire 80 columns may be used for data input.

FORTRAN PROGRAM EFFICIENCY HINTS

Same mode variables and constants in an arithmetic expression.

Reduced use of subscripts.

Constant subscripts rather than variable.

Arguments in common rather than calling list.

Non-varying values computed before entering DO loop.

FORTRAN ELEMENTS

Constants	Form	Examples
Integer	$n_1n_2...n_m$	2
	$1 \leq m \leq 18$	247
		314159265
Real	$n_1n_2...n_mE\pm exp_{10}$	3.14
	$1 \leq m \leq 15$	.0749
		314.E05
Hollerith	$nHf_1f_2...f_n$	7HG0GET1T
	$nLf_1f_2...f_n$	3HSON
	$nRf_1f_2...f_n$	10LABENDbbbbb
	$1 \leq n \leq 10$ for a replacement statement; $1 \leq n \leq 136$ for a format statement; maximum 19 continuation lines for a DATA statement	1RP 5HTHANX
	f=alphanumeric character	
Logical	.TRUE. .T.	-1, Integer
	.FALSE. .F.	All zero bits

FORTRAN ELEMENTS

Subscripts

The DIALOG data base stores only elemental variables and single subscripted arrays. Multidimensional arrays must be stored as single subscripted arrays. The following information is useful for referring an element of a multidimensional array as an element of a single subscripted array.

For DIMENSION A(L,M,N) the location of A(i,j,k) with respect to first element of A is:

```
(i-1+L*(j-1+M*(k-1)))*E
```

E is the number of words occupied by each element of A.

A subscript may be:

Integer constant

Simple integer variable

Arithmetic expression, not complex or double precision

Examples:

```
(1,J)
(1+3,J+3,2*K+1)
(3*K*1+3)
```

FORTRAN STATEMENTS

Subprogram Statements

PROGRAM s ($f_1 f_2, \ldots, f_n$)

SUBROUTINE s ($a_1 a_2, \ldots, a_n$)

FUNCTION f ($a_1 a_2, \ldots, a_n$)

ENTRY s

BLOCK DATA

EXTERNAL $v_1, v_2, \ldots, v_n$

END

Inter-Subroutine Statements

CALL s ($a_1 a_2, \ldots, a_m$)

RETURN

FORTRAN STATEMENTS

Data Declaration and Storage Allocation

REAL $v_1, v_2, \ldots, v_n$

INTEGER $v_1, v_2, \ldots, v_n$

LOGICAL $v_1, v_2, \ldots, v_n$

DIMENSION $v_1(i_1), v_2(i_2), \ldots, v_n(i_n)$

COMMON/x_1/a_1 .../x_n/a_n

EQUIVALENCE (k_1 , k_2),...,(k_{n-1}, k_n)

DATA $k_1/d_1/, k_2/d_2/, \ldots, k_n/d_n/$

Intra-Program Transfers

GO TO k

GO TO i,($k_1, k_2, \ldots, k_n$)

GO TO ($k_1 k_2, \ldots, k_n$),e

IF(e)k_1, k_2, k_3

IF(e)k_1, k_2

IF(e)s

Miscellaneous Program Controls

ASSIGN k TO i	PAUSE	STOP
CONTINUE	PAUSE n	STOP n

Loop Control

DO n i=m_1, m_2, m_3

FORTRAN STATEMENTS

Input/Output

I/O Control Statements

READ f,k	WRITE(u)k	PRINT f,k
READ(u)k	WRITE(u,f)	PUNCH f,k
READ(u)	WRITE(u,f)k	BUFFER IN(u,k)(A,B,)
READ(u,f)k		BUFFER OUT(u,k)(A,B,)
READ(u,f)		

Internal Conversion

ENCODE(n,f,A)k

DECODE(n,f,A)k

Tape Handling

ENDFILE u

REWIND u

BACKSPACE u

Miscellaneous

NAMELIST/$Y_1/a_1/Y_2/a_2\ldots/Y_n/a_n$

Format Declaration

FORMAT($q_1t_1z_1t_2z_2\ldots t_nz_nq_2$)

- q series of slashes (optional)
- t field descriptor or groups of field descriptors
- z field separator
- n may be zero

FIELD DESCRIPTORS

- srEw.d single precision floating point with exponent
- srFw.d single precision floating point without exponent
- srGw.d single precision floating point with or without exponent

FORTRAN STATEMENTS

Format Declaration

FIELD DESCRIPTORS

rIw	decimal integer conversion
rLw	logical conversion
rAw	Alphanumeric conversion
rRw	Alphanumeric conversion
rOw	Octal integer conversion
$nHh_1h_2...h_n$	Hollerith character control
nX	Intraline spacing
...	Hollerith string delimiters
/	Format field separators and formatted records demarcation

Printer Carriage Control

Character in first column	Resulting PRINT Operation
0	Double space after printing
1	Eject page before printing
+	Suppress spacing after printing
Blank or other than above	Single space after printing

FORTRAN FUNCTIONS

Intrinsic Function		Argument/Function
ABS(x)	Absolute value	Real/real
AINT(x)	Truncation, integer	Real/real
AMAX0(i_1,i_2,...)	Maximum argument	Integer/real
AMAX1(x_1,x_2,...)		Real/real
AMIN0(i_1,i_2,...)	Minimum argument	Integer/real
AMIN(x_1,x_2,...)		Real/real
AMOD(x_1,x_2)	x_1 modulo x_2	Real/real
FLOAT(i)	Conversion	Integer/real
IABS(i)	Absolute value	Integer/integer
INT(x)	Conversion	Real/integer
IFIX(x)	Conversion	Real/integer
INT(x)	Truncation	Real/integer

FORTRAN FUNCTIONS

Intrinsic Function		Argument/Function
ISIGN(i_1,l_2)	Sign$_i$ i_2 times i_1	Integer/integer
MAXO(i_1,i_2,...)	Maximum argument	Integer/integer
MAXI(x_1,x_2,...)		Real/integer
MINO(i_1,i_2,...)	Minimum argument	Integer/integer
MINI(x_1,x_2,...)		Real/real
MOD(i_1,i_2)	i_1 module i_2	Integer/integer
SIGN(x_1,x_2)	Sign x_2 times x_1	Real/real

External Function		Argument/Function
ACOS(x)	Arccosine	Real/real
ALOG(x)	Natural log	Real/real
ALOG10(x)	Log base 10	Real/real
ASIN(x)	Arcsine	Real/real
ATAN(x)	Arctangent	Real/real
ATAN2(x_1,x_2)	Arctangent x_1/x_2	Real/real
COS(x)	Cosine	Real/real
EXP(x)	e to the xth power	Real/real
SIN(x)	Sine	Real/real
SQRT(x)	Square root	Real/real
TAN(x)	Tangent	Real/real

APPENDIX G EXCLUDED NAMES FOR DATA BASE VARIABLES

Generally, the user of the DIALOG Executive System is free to specify names for data base variables and arrays. However there are certain names which are excluded. The excluded names are those used by the DIALOG executive system for option specification and by the analyst for 'DESIGN identifiers.' Certain other miscellaneous names are excluded.

DIALOG Executive System Options

BUILD	Option for dynamic construction of the data base.
DIVIDE	Symbol used for divide (/) in the operator directory.
DOLLAR	Symbol used for DOLLAR ($) in the operator directory.
ELTIME	Total elasped simulation time used in timing option.
EQUAL	Symbol used for equal (≐) in the operator direction.
EXPON	Symbol used for exponentiation (**) in the operator directory.
MINUS	Symbol used for subtraction (-) in the operator directory.
MLTPLY	Symbol used for multiplication (*) in the operator directory.
NOTEQL	Symbol used for a data base delimiter (≠) in the operator directory.
INDUMP	Option to print the modified input for every program
OUTDUMP	Option to print the special data base output file from each program.
PAGDMP	Option to print the internal string processing information
PLUS	Symbol used for addition (+) in the operator directory.

DESIGN Identifiers

The specification of a DESIGN identifier by the control directive:

'DESIGN identifier'

results in the identifier being stored in the data base. Once used the identifier is excluded from use as a data base variable name.

Miscellaneous Exclusions

Generally, the names listed above are the only exclusions the analyst need be concerned with. However, it is recommended that the communication command names and control directive names be excluded also.

ADD	Add command.
DELETE	Delete command.
DEFINE	Define command.
.	comment 'command'
CREATE	Create directive.
DESIGN	Design directive.
EXECUTE	Execute directive.
IF	If directive.
LOOP TO	Loop To directive.
RESTART	Restart directive.
UPDATE	Update directive.

☆U S GOVERNMENT PRINTING OFFICE. 1973-739-152/35

"The aeronautical and space activities of the United States shall be conducted so as to contribute . . . to the expansion of human knowledge of phenomena in the atmosphere and space. The Administration shall provide for the widest practicable and appropriate dissemination of information concerning its activities and the results thereof."

—NATIONAL AERONAUTICS AND SPACE ACT OF 1958

NASA SCIENTIFIC AND TECHNICAL PUBLICATIONS

TECHNICAL REPORTS Scientific and technical information considered important, complete, and a lasting contribution to existing knowledge.

TECHNICAL NOTES. Information less broad in scope but nevertheless of importance as a contribution to existing knowledge

TECHNICAL MEMORANDUMS· Information receiving limited distribution because of preliminary data, security classification, or other reasons. Also includes conference proceedings with either limited or unlimited distribution.

CONTRACTOR REPORTS· Scientific and technical information generated under a NASA contract or grant and considered an important contribution to existing knowledge.

TECHNICAL TRANSLATIONS: Information published in a foreign language considered to merit NASA distribution in English.

SPECIAL PUBLICATIONS. Information derived from or of value to NASA activities. Publications include final reports of major projects, monographs, data compilations, handbooks, sourcebooks, and special bibliographies.

TECHNOLOGY UTILIZATION PUBLICATIONS: Information on technology used by NASA that may be of particular interest in commercial and other non-aerospace applications. Publications include Tech Briefs, Technology Utilization Reports and Technology Surveys.

Details on the availability of these publications may be obtained from:

SCIENTIFIC AND TECHNICAL INFORMATION OFFICE

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Washington, D.C. 20546